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.**

I. Introduction

EVERY 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 still subject to manual inspection and SQL statement updates for each subsequent month. The risk of errors and inaccuracies resulting from missed variations along with null values from a manual row-by-row review remains. A better 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. The relational database MySQL was chosen primarily because it is free, open source, and also because it offers the relational database architecture with a readily available user interface similar to Company X's SQL Server. The programming language Python was chosen not only for its simple programming syntax, but also for its widespread use which will ensure that any additions/modifications that would need to be made in the future would be simple. Python has been shown to be viable as a programming language to resolve real-world data cleaning problems like our project. All of these make coding a solution easier, faster, and more 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 original database schema. The solution also replaces 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.

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).

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 two tables serve as references to compare new vendor name values to data collected in subsequent months. This solution provides an automated data cleaning utility.

II. 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.

In the provided data set there are seventy-one (71) non-standardized variations of 'AMERICAN AIRLINES' with the volume growing with each subsequent month's entries. While the data disparity may 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.

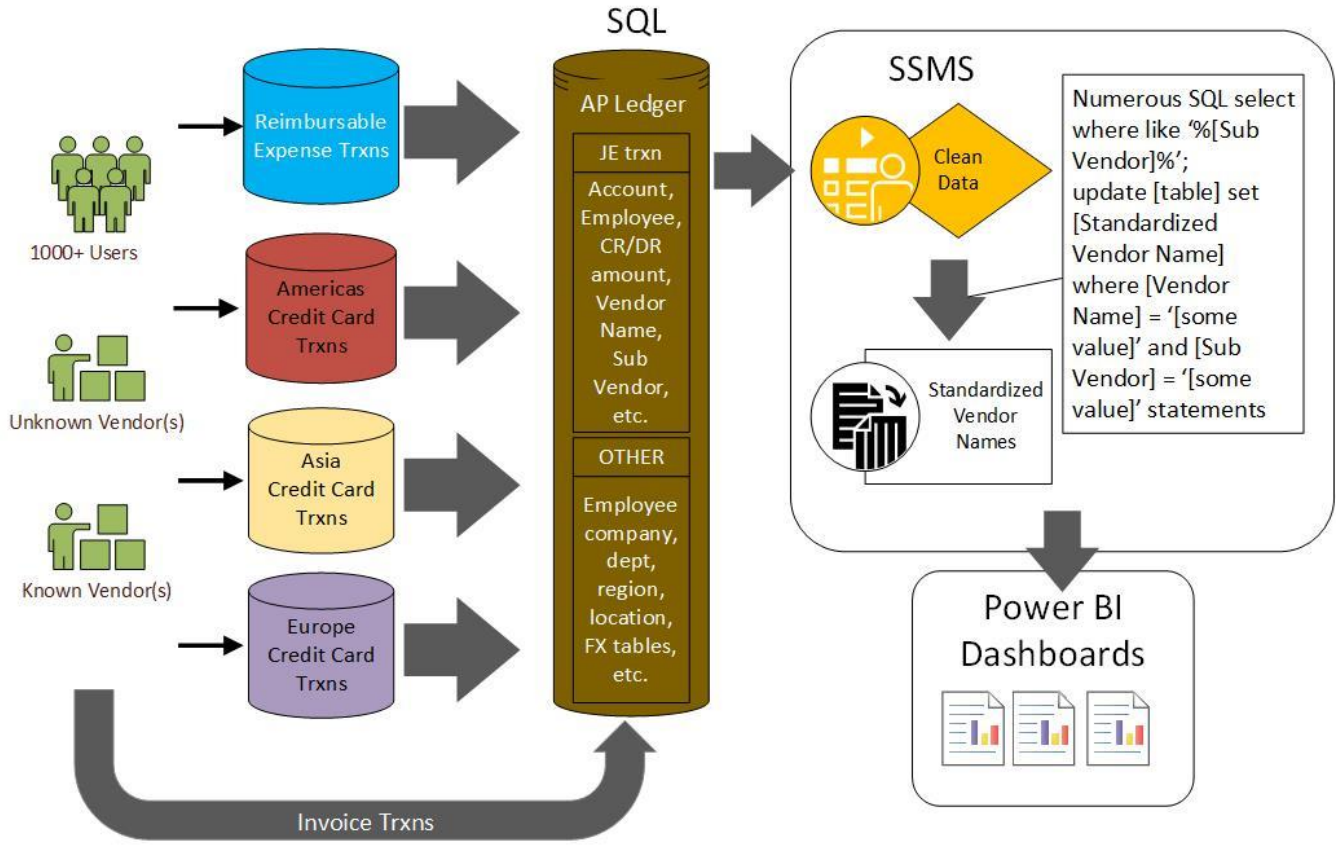

Fig. 1. Company X's travel and reimbursable expense entries' data cleaning process.

TABLE I
VARIATIONS TO BE EVALUATED FOR 'AMERICAN AIRLINES'

VendorName	SubVendor	StandardizedVendor
CONCUR UPLOAD	American Airlins	AMERICAN AIRLINES
CONCUR UPLOAD	American Airlnes	AMERICAN AIRLINES
CONCUR UPLOAD	Amarican Airlines	AMERICAN AIRLINES
CONCUR UPLOAD	American Airlines Food	AMERICAN AIRLINES
BCD Travel USA LLC	AMERICAN AIRLINES 800-433-7300 TX	AMERICAN AIRLINES
BCD Travel USA LLC	AMERICAN AIRLINES DALLAS TX	AMERICAN AIRLINES
BCD Travel USA LLC	AMERICAN AIRLINES IN IRVING TX	AMERICAN AIRLINES
BCD Travel USA LLC	AMERICAN AIRLINES PHOENIX AZ	AMERICAN AIRLINES
BARCLAYCARD COMMERCIAL	AMERICAN AIRLINES	AMERICAN AIRLINES
Bambora	American Airlines	AMERICAN AIRLINES
American Express Services Europe Ltd (GBP)	AMERICAN AIRLINES	AMERICAN AIRLINES
American Express Services Europe Ltd (GBP)	American Airlines In BCD TRAVEL IREL	AMERICAN AIRLINES
American Express Services Europe Ltd (GBP)	AMERICAN AIRLINES THE TRAVEL COMP	AMERICAN AIRLINES
American Express Services Europe Ltd (EUR 3753)	AMERICAN AIRLINES	AMERICAN AIRLINES
American Express International Inc.	AA HONG KONG MCCY HKD BSP	AMERICAN AIRLINES
American Express International (Japan)	American Airline	AMERICAN AIRLINES
AMERICAN EXPRESS (ACT NO 31008)	AA ANCILLARY SALES	AMERICAN AIRLINES
AMERICAN EXPRESS (ACT NO 31008)	AA ARC	AMERICAN AIRLINES
AMERICAN EXPRESS (ACT NO 31008)	AA RESERVATIONS SALES	AMERICAN AIRLINES
AMERICAN EXPRESS (ACT NO 31008)	American Airlines	AMERICAN AIRLINES
AMERICAN EXPRESS (ACT NO 31008)	AMERICAN AIRLINES 800-433-7300 TX	AMERICAN AIRLINES
AMERICAN EXPRESS (ACT NO 31008)	AMERICAN AIRLINES DALLAS TX	AMERICAN AIRLINES
AMERICAN EXPRESS (ACT NO 31008)	AMERICAN AIRLINES DIRECT	AMERICAN AIRLINES
AMERICAN EXPRESS (ACT NO 31008)	AMERICAN AIRLINES IN IRVING TX	AMERICAN AIRLINES
AMERICAN EXPRESS (ACT NO 31008)	AMERICAN AIRLINES INC	AMERICAN AIRLINES
AMERICAN EXPRESS (ACT NO 31008)	AMERICAN AIRLINES PHOENIX AZ	AMERICAN AIRLINES
AMERICAN EXPRESS (ACT NO 31008)	AMERICAN AIRLINES WEB SAL	AMERICAN AIRLINES
AMERICAN EXPRESS (ACT NO 11005)	AA ANCILLARY SALES	AMERICAN AIRLINES
AMERICAN EXPRESS (ACT NO 11005)	AA ARC	AMERICAN AIRLINES
AMERICAN EXPRESS (ACT NO 11005)	AA RESERVATIONS SALES	AMERICAN AIRLINES

For this case study, Company X provided 1,009,303 tuples from the last two calendar years 2017-2018. This data was provided to help determine whether a solution designed and executed with Python and MySQL can provide a better method to clean and standardize these vendor names. The metric for this decision, while addressed in the Test Results section, boils down to whether the end solution is more efficient in terms of time taken for the task and also in what specialized manpower is required.

III. SOLUTION DESIGN

A. Solution Tool Selection

For the setup of the solution to be easily replicable, a package management and deployment system called 'Anaconda' was used to manage the installed libraries and packages of Python including 'pandas', 'mysql.connection', and 'difflib'. 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. More importantly it has a readily available user interface with similar capabilities to Company X's SQL Server. MySQL's quick start capability from software download to complete installation regardless of underlying platform and the inclusion of a comprehensive set of migration tools sufficiently enables a test environment that simulates Company X's data model. The programming language Python was chosen not only for its simple programming syntax, but also for its widespread use which will ensure that any additions/modifications that would need to be made in the future would be simple. Also, Python's code readability which is due to it being a high-level programming language makes coding a lot easier and more efficient in manipulating Company X's data in this initial solution. Finally, Python was also chosen because of its viability as a programming language to resolve real-world

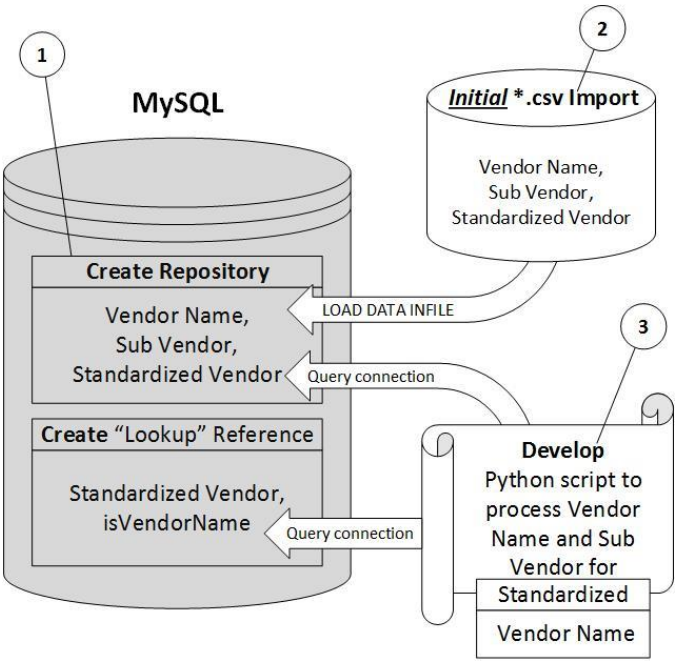

Fig. 2. Initial Solution Design

database problems like our project. Python's library 'pandas' was selected due to seeing the practical application and the academic literature involving its ability to solve the business problem of cleaning data. The use of 'mysql.connection' to connect MySQL and Python was chosen solely because it is officially supported by MySQL which will help ensure that it does not become an obsolete library.

B. Refinement

After the initial importing of Company X's data into MySQL, Fig. 2. Initial Solution Design depicts the basic workflow of the solution to create a reference "lookup" table to import all the distinct valid combinations from previously standardized data into the table, then develop and use a Python script that references the "lookup" table to automatically populate the valid entries into the standardized vendor name field.

Serious insight was gained in exploring the differing ways to import the data into MySQL. Using the Table Data import wizard was first attempted. While intuitive and user friendly, it was incredibly slow, taking about a ½ sec for each row. This method was aborted because of the 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 settings is the CHARACTER SET in the data import to make sure that the database will support the appropriate languages for this dataset.

A significant detail learned in the process of using the 'mysql.connection' library to connect Python and MySQL 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 utilizing the current connector will not need to be altered because the query is executed through the connection object. For example, if it is necessary to connect to the Microsoft SQL Server database instead of MySQL, only the connector has to be modified rather the underlying code.

For manipulating the database schema, a 'cursor' was used to execute the queries. As there are many different types of cursors, it was determined that a "Buffered" cursor could best serve the objective to iterate through the records. However the actual act of retrieving the data does not require a cursor. This is because the 'pandas' library, which was chosen because it contains data structures and data analysis tools, also has a method for selecting and returning data from a database into a dataframe without a cursor.

C. Final Design and Execution

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 which we dubbed the 'mistake' table and only import the Sub Vendor values with the logic to use the Vendor Name if the original Sub Vendor value is null. We added a function from the 'difflib' library to determine how similar one value is to another. This value is called a 'ratio' which is a value between '0' and '1', inclusive. If a ratio equals '1' the two values are a perfect match. This value is the key for checking un-standardized values against the ones 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 (as a foreign key) paired 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.

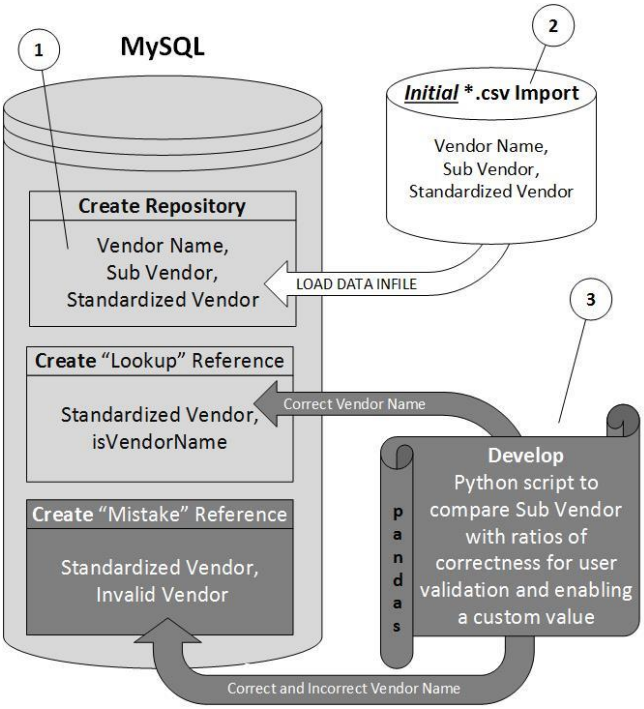

Fig. 3. Final Solution Design

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 decided to use the SQL clause 'LIMIT' to pull in specific rows of un-standardized data from the original 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 clause allows the management of how much data can be imported into Python at a time.

```
======================================================
UNITED AIRLINES IRVING TX
------------------------------------------------------
c          : Create Custom Value
<num>      : Choose row # <num>
<other>    : Anything else to see next set of rows
------------------------------------------------------

   Ratio          LookupVal                      MistakeVal
0  0.818    UNITED AIRLINES             UNITED AIRLINES INC
1  0.769  AMERICAN AIRLINES     AMERICAN AIRLINES IRVING TX
2  0.582    HAINAN AIRLINES  HAINAN AIRLINES HOLDING CO LTD
3  0.578    DELTA AIR LINES            DELTA AIR LINES INC.
4  0.571  EMIRATES AIRLINES                            None


 Choice:
```

Fig. 4. Main User Interface

Development of a simple user interface shown in Fig. 4. 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. In the event that there is a perfect match (ratio = 1) the script automatically inserts into and updates the 'lookup' and 'mistake' tables and continues on to the next value. It does this without any prompting or interaction with the user. With the ratio being any value other than 1 the user can select one of the suggested matches (which are sorted in descending order by ratio) or create a custom value if the function did not return a satisfactory match. The user is then prompted whether to keep or retype the custom value. When the user creates the custom value it is compared to all the ones in the 'lookup' and 'mistake' tables via the creation of another ratio to allow the

user to either use one from those tables, or to actually use the custom value. If the custom value is used, then it is inserted in both the 'lookup' and 'mistake' tables to allow for future comparison. Otherwise the custom value is discarded when the user selects one of the values that already exist within the 'lookup' table. Finally, whenever the desired value is selected automatically by the user from the 'lookup' table, 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.

IV. Test results

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

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

Of the starting 1,009,303 rows of journal lines comprised of 22,513 unique vendor name values, we captured 970 correct reference values and 1,754 mistake reference values during the process of the building and testing of 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 with them being frequently lower than 0.6 with progressive increase to 0.8 for some values. This is understandable because the reference values inside the 'lookup' and 'mistake' tables need to be built up so there are more exact and close matches to the new values that are compared. Thus, the tedium in building these initial reference tables remains time-consuming and labor-intensive. However, we are optimistic about the resulting time saved by cleaning data like this once we have an increased volume of reference in those two tables.

This solution design eliminated the need to write SQL select and update statements to correct the vendor name disparities. By replacing it with a Python-based, non-technical user interface, we created the ability for non-technical users to immediate benefit by allowing them to do the data cleaning process.

Furthermore, as mentioned in the background, these vendor name references extend across all eighty-two travel and reimbursable expense types. With this solution Company X is no longer limited to standardizing the values of the higher spend expense types.

V. Conclusions

The primary conclusion we have reached from this case study is that it is indeed useful to use Python with the 'pandas', 'mysql.connection' and 'difflib' libraries 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 interchangeably switched out provides flexibility in reading and writing to different databases. We have to be careful to sanitize SQL queries that contain special characters as such as quotation marks when concatenated strings Python.

We also conclude that it is feasible to modify a production database schema in resolving this problem despite cautionary tales. By modifying the current 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 are natural language processing, machine learning algorithms, and other commercial data quality tools that can be further explored and implemented to resolve Company X's data quality problem. These would mostly likely have an even greater efficiency albeit at a much higher purchase price. All of these alternate solutions hypothetically could be implemented within a Python and MySQL solution; however, the specialized knowledge mentioned above of natural language processing, machine learning algorithms, and data structures would most likely be required.

VI. References and Footnotes

A. References

[1] B. Shetty, "Natural Language Processing(NLP) for Machine Learning," *Towards Data Science*, 24-Nov-2018. [Online]. Available: https://towardsdatascience.com/natural-language-processing-nlp-for-machine-learning-d44498845d5b. [Accessed: 22-Apr-2019].

[2] C. Anderson, "How can I connect to MySQL in Python 3 on Windows?," *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/4960048/how-can-i-

connect-to-mysql-in-python-3-on-window. [Accessed: 22-Apr-2019].

[3] "Data cleansing," *Wikipedia*, 08-Apr-2019. [Online]. Available: https://en.wikipedia.org/wiki/Data_cleansing. [Accessed: 22-Apr-2019].

[4] "difflib - Helpers for computing deltas," *difflib - Helpers for computing deltas - Python 3.7.3 documentation*. [Online]. Available: https://docs.python.org/3.7/library/difflib.html#sequence matcher-examples. [Accessed: 22-Apr-2019].

[5] G. Gulipalli, "5 Steps to Data Cleansing of Customer Data," *Invensis Technologies*, 10-Jul-2018. [Online]. Available: https://www.invensis.net/blog/data-processing/5-steps-data-cleansing-customer-data/. [Accessed: 22-Apr-2019].

[6] "How To Change Column Names and Row Indexes in Pandas?," *Python, R, and Linux Tips*, 01-Mar-2019. [Online]. Available: https://cmdlinetips.com/2018/03/how-to-change-column-names-and-row-indexes-in-pandas/. [Accessed: 22-Apr-2019].

[7] M. B. M. Bhasi, "How should I tackle --secure-file-priv in MySQL?," *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/32737478/how-should-i-tackle-secure-file-priv-in-mysql. [Accessed: 22-Apr-2019].

[8] P. Norvig, "How to Write a Spelling Corrector.", [Online]. Available: https://norvig.com/spell-correct.html. [Accessed: 22-Apr-2019].

[9] T. Barrus, "pyspellchecker," *PyPI*. [Online]. Available: https://pypi.org/project/pyspellchecker/. [Accessed: 22-Apr-2019].

[10] S. Lynn, "Using iloc, loc, & ix to select rows and columns in Pandas DataFrames," *Shane Lynn*, 09-Jul-2017. [Online]. Available: https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/. [Accessed: 22-Apr-2019].

[11] "Why Data Scientists Love Python," *CBT Nuggets*. [Online]. Available: https://www.cbtnuggets.com/blog/2018/09/why-data-scientists-love-python/. [Accessed: 22-Apr-2019].

[12] D. Huynh and S. Mazzocchi, "Welcome!," *openrefine.github.com*. [Online]. Available: http://openrefine.org/. [Accessed: 22-Apr-2019].