Handling Different Measurement Units Across Multiple Data Sources Using CSV Files

Mihran Simonian - 12386294
mihran.simonian@gmail.com
University of Amsterdam

ABSTRACT

While the manufacturing industry attempts to implement open data communication over the whole supply chain, existing computer (ERP) systems are maintained in-place at individual manufacturing plants. The wide range of systems, together with the wide range of sensors used to measure manufacturing parameters, create multiple challenges. As supply chains operate and procure more cross-boarder than ever before, there is a growing need for a standardized units of measurements and other key parameters (such as ordering quantities). However, computer systems are hardly designed to cope with units, let alone when introducing multiple units. In this paper I will analyze the existing methods through the means of testing and will design and propose a system to unify these units.

KEYWORDS

Supply chain, units, convert, databases

1 INTRODUCTION

The manufacturing industry is currently going through a major development, described as 'Smart Industry 4.0'[5]. By using computers, big data and the implementation of various sensors, manufacturers attempt to optimize their production flows and lower costs, whilst increasing reliability and optimize production speeds.

Simultaneously OEM producers (manufacturers whom combine loose parts to assemble one product) attempt to optimize their supply chain, by combining the data streams of various suppliers attempt to optimize it's supply chain, in order to avoid production halts when a disruption occurs in the supply chain, as production halts are very costly and can create significant delivery delays to the end-users.

As measurement units are different across countries, unit conversion is required and must be implemented before automatic ordering can take place. However, despite many efforts, the various current methods all lack in one aspect or another, as elaborated on in the appendix to this paper.

Global Organization

Countries around the world start with research and innovation programs, in which supply-chain management techniques are attempted to be improved using data acquired locally and globally[4].

Due to the fact that production facilities can possibly be spread all over the world[1], there is an increasing issue that is rising but currently less highlighted as computer systems are currently not necessarily designed to cope with multiple measurement units[2].

Physical Units

Differences in measurement units used in manufacturing industry are visible on all type of fields, yet temperatures, forces, distances, volume and weight are the main types (we can derive most others from these types). To illustrate the magnitude of the issue; in 1999 NASA lost one of it's satellites orbiting the planet Mars, due to a programming error in which a US Customary unit was implemented but the Metric unit had to be used instead[2][1]. Surely manufacturing a car or assembling parts for a lithography machine[3] will not result in the same financial (and social) consequences, but it is an indication that we should not discard the issue and it's potential consequences.

2 RELATED WORK

Unit Conversion

The issue of using multiple sensors and vastly storing these in computer systems has already been highlighted in previous work [7], in which data variable types were highlighted as possible sources of conflict. This was in 1990, however in 2013 there was still reason enough for concern, as [2] described in his paper.

Integrating ERP Systems

Co-currently the Dutch research institute TNO is currently developing (test version live since January 2020) a communication protocol (labelled as SCSN) in cooperation with

[1]https://www.asml.com/en/company/sustainability/responsible-supply-chain
[2]https://mars.jpl.nasa.gov/msp98/news/mco991110.html
[3]https://www.asml.com/en/technology

various manufacturers[4][5]. This communication standard enables manufacturers to share various production parameters (data) in an easy manner, whilst the individual manufactures remain to be able to use their own ERP[6] system, which is preferred by manufacturers due to various reasons (legacy purposes, proprietary software, vendor lock-ins, etc.). The SCSN protocol is introduced as it assures that companies do not need to 'give up' their own dedicated ERP systems, whilst being enabled to share data with other manufacturers (cross ERP). Thus, systems might communicate via live data connections or by means of backlogs, where on a daily night backlogs are loaded into systems (by using csv file containers for instance).

Interviews conducted with the research department of TNO highlighted that upon implementation of ERP systems one of the main considerations is measurement units in general. The correct implementation of units inside the data sharing facility is such a key aspect, as not only the units themselves can differ, but the way that the same unit is used in a system can also differ; the unit representation. The research done by TNO highlights the importance of unit conversion, but also understand what the unit stands for.

Python Libraries

Unit Conversion. Unit conversion is not a new thing, even the basic Windows calculator is able to convert units. Multiple libraries for Python[7] have been written, such as PintPy[8] or the 'unit-converter' from PyPi[9]. Pypi actually hosts many libraries for unit conversion, hereunder a subset of library packages performing unit conversion, all named very equally:

- Unitconvert[10]
- Unit-convert[11]
- Unit-converter[12]
- Unit-conversion[13]

The above list is slightly confusing as to how the name giving has taken place. Especially when writing code one could easily confuse the packages, as actually happened to myself upon making this report. The names are so similar, that also referring to online manuals can lead to the wrong solution, without you as a programmer realizing that you are actually looking at the wrong website. To add to the confusion all libraries have their own specific way of dealing with unit conversion. Some require additional data subsets to be added, some in one way some in another way further complicating implementation. This all makes it quite confusing to what can be expected and how the data needs to be given in order to get it converted in the right way.

Complexity

The essence of unit conversion is actually not a hard concept; you take a number and then convert it according to the formula which is a known static variable (or formula, as we will see later in this paper). The issue is that there are many units, and they can all occur between each other within ERP systems. When automatically submitting an order from company A to company B, this might lead to all sorts of issues. It becomes apparent that this is (one of the main) reason(s) why various initiatives (such as SCSN) are struggling with up-scaling their potential market, as companies are not fully in trust of such a system (yet).

3 RESEARCH

The framework of this research is discovering the current status of the Python libraries and propose an alternative solution. The ERP systems are hypothesized to be csv files which will be combined, similar to what the SCSN network is attempting to achieve.

Aim

This research is aimed to highlight the importance of identifying the need to convert a wide range of varied units, represented in various data sources when combining these sources. By displaying the limitations of various Python library packages, we will discover the current status of Python libraries when it comes to unit conversions. An alternative, own proposed method will also be designed.

Relation to Big Data: Variety

The angle of approach for the current paper is designed around the 'variety' aspect of the Big Data set of V's. The question how to reduce variety of units, which can potentially lead to all sorts of practical problems (losing a satellite in space) is key and as a result we will also notice what impact all these various units (together with the usage of the library packages) will have on translation speeds. This leans a bit towards the V of 'velocity', a logical consequence as more variety leads to less available computation power. The main focus however remains to be on 'variety'.

Relation to Data Science Methods

Data scientist often use statistical methods in order to test certain hypothesis, such as calculating the mean or median. Especially in the field of data science, variations in numbers

[4]https://smartindustry.nl/fieldlabs/8-smart-connected-supplier-network
[5]https://www.brainportindustries.com/nl/berichten/maakindustrie-aan-de-slag-met-digitalisering
[6]https://en.wikipedia.org/wiki/Enterprise_resource_planning
[7]https://www.python.org/
[8]https://pint.readthedocs.io/en/0.11
[9]https://pypi.org/project/unit-converter
[10]https://pypi.org/project/unitconvert/
[11]https://pypi.org/project/unit-convert/
[12]https://pypi.org/project/unit-converter/
[13]https://pypi.org/project/unit-conversion/

can push the research into the wrong direction. By removing a fundamental reason why numbers can differ (different units) we can further improve the field of data science and improve future researches.

Research Question

During this research I would like to answer the following question:

- How do we combine various units and translate them into a single representative unit system?

The following sub questions will also become part of the research:

- What is the current level of the python libraries?
- What solutions can be applied?
- How is the implementation of current existing solutions?
- Which pre-cleaning steps are required?

4 METHOD

Data Sources

For this research I will use a self-generated dataset upon which the introduced method will be tested. The dataset is generated using Microsoft Excel[14] as it enables us to generate random numbers rather quickly, and convert them to a set of datatypes.

Filetype

For the current implementation the popular csv format[15] is used. The csv format can be used on both Windows and Unix based systems and is recognized by most database software packages. Famous ERP packages such as Baan, SAP and Oracle also accept csv formats. The popularity does not only end there, as the Pandas library also supports csv files. The only negative side is that csv files do not support live-transmission of data, as they are export files of the complete database (thus they first need to be generated). However for the goal of this research this is not important, as the focus lies on unit conversion.

Unit Types

For the current research I will solely focus on the application of the following units:

- Temperatures
- Mass
- Distance

There is sufficient differences in these units in order to draft up a complex study, highlighting the differences of unit systems and complexities involved in converting them.

One Unit System

The proposed solution will align all units to be represented in one unit system; this is the international recognized SI system[16].

Python

The proposed solution is designed using the Python language[17] is used as it allows us to work with various libraries as described under the relevant work section.

IDE. The program will be written in the IDE[18] Visual Studio Code[19] using the add-in package[20] which provides additional usage of Python within Visual Studio Code. By using the Jupyter Notebooks[21] format, a interactive interpreter is created which is very suitable to quickly build programs. Support for Jupyter is integrated in Visual Studio Code upon installation of the Jupyter environment on the computer system.

Libraries. The following Python libraries will be continuously used in the testing and when designing a solution:

- Pandas[22]

Testing Existing Libraries

By testing how a standard csv file can be imported and how units in the csv file can be converted, it will become clear what the current status quo is.

The following Python libraries will be tested:

- PintPy[23]
- PiPy: Unit-convert[24]
- PiPy: Unit-converter[25]

Each library will be evaluated on the following points:

- How does it work?
- Did any error occur?
- Solutions of error prevention and consequences
- Conclusion

Please refer to the appendix for an elaborate analysis of these libraries, how they work and what limitations there are.

[14] https://www.microsoft.com/en-us/p/excel/cfq7ttc0k7dx?activetab=pivot%3aoverviewtab
[15] https://en.wikipedia.org/wiki/Comma-separated_values
[16] https://en.wikipedia.org/wiki/International_System_of_Units
[17] https://www.python.org/
[18] https://en.wikipedia.org/wiki/Integrated_development_environment
[19] https://code.visualstudio.com/
[20] https://marketplace.visualstudio.com/items?itemName=ms-python.python
[21] https://jupyter.org/
[22] https://pandas.pydata.org/
[23] https://pint.readthedocs.io/en/0.11/
[24] https://pypi.org/project/unit-convert/
[25] https://pypi.org/project/unit-converter/

This includes actual Python code, in case the reader wants to implement these libraries themselves.

Design Own Method

As the libraries are limited in their own way, I will also discuss a proposed solution to work around the limitations. In essence I will design an alternative solution to performing unit conversion using Python.

Scope and Limitations

This research is a mere introduction to the vibrant world of (measurement) units and is intended to highlight the importance of unit alignment. There are for instance differences between UK (Imperial) and US (Customary) units[26], even though they use very similar notations (to add to the confusion). I will however not dive extensively deep into this topic as it does not add much to the model itself, it's a mere iteration of an existing situation and thus 'just another conversion'.

Furthermore this research will not try to optimize and reduce the computational cycles as required in order to transform units, as this would transform the research more into reviewing this unit issue from the big data perspective of 'velocity' oppose to the intended 'variety'.

5 IMPLEMENTATION

Requirements

The self generated dataset will contain the measurement parameters and unit representation inside the dataset. This allows us to build datasets quickly, and allows normal users to change the unit quickly in case someone types the wrong unit accidentally. This also clarifies the unit in place and displays what the data represents.

We also want our database solution to work dynamically, as we are connecting databases. Normally we would not change units in our databases quickly, however this potentially can occur, especially if we were to design a centralized system, similar to the SCSN network (which works as a 'translator' between two companies, and thus is faced with different units from time to time).

Own method

Below is my proposed solution. I have followed the main questions as with the other libraries and have added additional information afterwards, in which I highlight the specifics which need to be taken into account when dealing with various number formats, data types and unit conversions.

How does it work? By importing the csv files, we can fill up a pandas *DataFrame* [27] tabulation rapidly. This imported data is not useful as long as we do not know what it represents. This is recorded in the 'metadata', which describes for instance 'this column has temperatures in Celsius'. The proposed method requires datasets to contain this information explicitly, which can than be imported and automatically will populate the 'metadata' of the pandas DataFrame. As the databases are in csv format, which can be read and edited by most software, this system allows a user-friendly solution, where normal users are also able to understand (and adjust if required) what is understood to be represented in a database by the computer program.

Did any error occur? Initially yes, as not all units are simple conversions (multiple, divide) it was not as simple to simply use one numerical amount. Furthermore no real issues were encountered.

Solutions of error prevention and consequences. By using a lambda [28] function in a dictionary I was able to solve the calculation of Celsius - Fahrenheit.

Conclusion: Benefits over using existing libraries. The proposed solution designed by author has a few benefits over the libraries as tested previously. The main benefit is the introduction of being able to import customize-able dictionaries

Using a dictionary for the application of unit conversions

By using a dictionary we can make a 'dynamic' work space, in which we can set unit conversions whilst not interfering with the actual code. This can be handy when we want to adjust the field of units we want to use and we would like novice users to be able to adjust these as well.

Unit conversion numbers and formulas

As described previously not all conversions are (unfortunately) simple multiplications or divisions. We have to sometimes fill in complete formulas. Please refer below to the dictionary algorithm, which is able to solve both formulas and 'simple' conversions. The line numbers 7 to 9 display such showcases (they use the function lambda). In the appendix the actual Python code can be found for this algorithm.

Customizable Unit Transform Dictionaries. Programming is a specialty not managed by everybody. By implementing multiple functions which can import and use a customize able translation lists, we allow people who do not master

[26] https://en.wikipedia.org/wiki/Comparison_of_the_imperial_and_US_customary_measurement_systems

[27] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

[28] https://en.wikipedia.org/wiki/Anonymous_function

Algorithm 1: Convert Units using multiplication or formulas

```
i = importvalue
if i not in dictionary then
    display errormessage
else
    x = i from dictionary
    if x is number then
        multiply x with data
    else
        apply formula x to data
    end
end
```

programming to also use the software. The benefit of this, is that we can tailor make translations lists of units, in order to assure we only use those units which will actually occur.

The proposed system uses csv files for the import of unit lists. The benefit of this, is that many programs can read and edit csv files; such as windows notepad, but also powerful software like Microsoft Excel. These programs are very general used software packages, which are installed on many computers and thus this allows also non-programmers to check whether the implemented translation list is actually correct.

Algorithm 2: Import CSV dictionary

```
r = csvfile(delimiter =';')
dictionary = empty
for key, value in r do
    dictionary.add[key] = value
end
```

Pre-cleaning of data; assuring numeric data types, assure dots presented as commas, perform numeric conversions

Sometimes a number displayed on the screen is not actually represented in computer memory as a number. If we want to perform calculations on numbers, we need to be sure that the imported data actually contains number data types for the computer to be able to interpret them correctly. A function was designed which will clean up the input data:

- Checks input type
- Assure dots; '.'
- Strings get converted to floating points
- Returns float type or error

Check input type. The program identifies whether a Boolean datatype is detected, which can potentially occur in a product database (or technical documentation for that matter), for instance when we specify whether a certain future is included in the product design. Therefore this line has intentionally been added in this code.

In the future we could add more data types, such as lists, tuples or dictionary data types or other objects. This would assure a dummy proof solution, but this seems passing by the idea of the current exercise as this would be simply copy existing lines and does not add to the proposed method, but merely expands it to make it more dummy proof (but also less universal).

Assure dots; '. Dots and commas are common in numbers and can become confusing when comparing international number formatting. There are thousand separators, designed to make it easier for people to read numbers, but there are also decimal separators [29], which are part of a number. We are only interested in the decimal separator, as it really says something about the number (2,01 is something else than 201). It is therefore evident we do not simply throw away the dots and commas but convert them to the system we prefer.

Algorithm 3: Pre-clean data

```
x = data
if x is boolean then
    display boolean errormessage
else
    if x is string then
        x = x.replace( , to . )
        if . in x > 1 then
            remove all . except 1
        end
    end
    make x to float datatype
end
```

Versatility Through Simplicity. The real power of the proposed solution is the simplicity. By allowing dictionaries in the csv format to dictate the conversions of units, this solution allows users the freedom to adjust which conversions are really required. As the dictionary keys are required to align with the database headings of columns, this creates a double verification whether the correct unit is in place. By the definitions used in the dictionary, mistakes are reduced as the user has to write down each individual conversion as:

```
1    [subject] [unit_system] [unit]
```

[29] https://en.wikipedia.org/wiki/Decimal_separator

6 EVALUATION

Adding New Manufacturers

When adding new manufacturers we need to be careful with the implementation process. The proposed system relies heavily on the usage of dictionaries, as the system provides us the freedom to do as we wish and we are even in the possibility to produce separate dictionaries for individual suppliers, as the code has been written cleanly and allows us to do so (we can re-use the same functions for multiple dictionaries). Please refer to the appendix for the actual code.

Implementation process. The alignment of variables is a strict requirement upon implementation. This can be achieved by adjusting the metadata inside the csv files, as these describe what each column actually represents. The negative side is that this has to be done specifically for each client which is time intensive. This however is not necessarily bad, as it does assure clean and proper alignment of variables and their representations over all ERP systems.

Synchronizing Data

When combining various data sources we will run into synchronicity issues. A master has to control all versions, delays and what do we do when the master dies? It's a single node

7 DISCUSSION

Computer Imprecision

By design, computers are not able to understand numbers and what they mean. Therefore, computers are always slightly inaccurate. This is because of the difference between how humans calculate (using a 10 base number system), and how computers calculate (using a 2 number base system). This is 'caused' by the IEEE 754 standard [30] used deep inside the hardware technogloy in computers. Please refer to the below to see the effect:

```
# Notice the strange output
x = 0.1 + 0.2
print(x)
# Output:
0.30000000000000004
```

Importance to this database. This imprecision has its effect on the database, especially as we are converting numbers and need to be sure about what we read and it's correctness.

Solving Computer Number Imprecision. Dealing with this computer uncertainty creates additional programming requirements. By implementing additional code one can verify whether this is a neglect able difference. There are multiple solutions, such as using the 'decimal' datatype or compare differences where the result is calculated and compared to an arbitrary number.

The negative side is that this will require more computational counts, and thus reduces the system's processing speeds.

Human Factor

By using one unit system, we still don't prevent people from entering the wrong unit conversion. By using csv files (accessible through various office software packages) we do limit input locations, but it still allows the human factor to create potential hazards. An alternative solution is applied by the Python library package PintPy, which allows the numbers to represent a property in the programming code. This means that there is a 'check' by the computer code to verify whether the unit actually is aligned with the other unit. This should prevent these issues from arising, however as discussed in the test, PintPy comes with it's own negative sides.

Other Units Used in Supply Chain

When considering a supply chain of manufacturers we should not disregard the existence of other units than technical or physical units, which are important but more from a engineering perspective. Supply chain planning revolves mainly around allocating resources efficiently. Various researches [3][6] have focused on issues such as:

- Work shifts with various hourly rates
- Discount policies
- Production times
- Ordering quantities

The main point to take from this is that products or units can be reflected differently in various systems; when we order 'a box of screws Z' from company A instead of company B, the actual quantity included inside the box can differ, or the delivery time can differ to that we would have expected. An important point to address when integrating supply chains, as ordering items is one of the key tasks in supply chain management.

Different Representation of Units Across Variables. ERP systems can be a mess when looking at how the same unit is represented for different variables. From personal experience I know that the ERP system which my employer utilized (Baan IV[31]) contained different representations of the same unit for different parameters, such as with days and (again)

[30] https://standards.ieee.org/standard/754-2019.html

[31] https://www.xibis.nl/producten/baanivenv

package units. As an example; delivery times would be represented in calendar days whilst production times would be represented in work days. When integrating the various ERP systems, such differences cannot be ignored and have to be unified as well.

Existing Solutions for Workdays. To lessor extent Workdays are also a known issue in computer management. Excel offers the function 'workday'[32], for example. For Python there is a library for 'workdays'[33] as well, just as the package 'businesshours'[34]. These libraries are necessary when we want to calculate the 'earliest shipping time' as commonly used in the field of supply chain.

8 FUTURE WORK

Limitations of Current Market

The currently available methods allow for either standard units to be converted, e.g.; kilograms to grams, Fahrenheit to Celsius and so on. These are physical units and are important, but as mentioned previously, these are not the only units that are being used within manufacturing and supply chain management.

The 'workday' and 'businesshours' libraries as currently available on their part do not allow to convert units.

There is no research conducted yet in which these two important items can be combined into one, by applying both packages on a single tabulation, thus providing an incremental improvement for the overall research of combining the various ERP systems.

ACKNOWLEDGMENTS

To Hannes Mühlheisen, for the fun and joy during the lectures and for setting up this exercise, together with Cristian Rodriguez Rivero and Shuo Chen.

REFERENCES

[1] MCO Mishap Investigation Board. 1999. Mars Climate Orbiter Mishap Investigation Board Phase I Report November 10, 1999. https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf

[2] Marcus P Foster. 2013. Quantities, units and computing. *Computer Standards & Interfaces* 35, 5 (2013), 529–535. https://doi.org/10.1016/j.csi.2013.02.001

[3] Stephen C Graves, David B Kletter, and William B Hetzel. 1998. A dynamic model for requirements planning with application to supply chain optimization. *Operations Research* 46, 3-supplement-3 (1998), S35–S49. https://doi.org/10.1287/opre.46.3.S35

[4] Boudewijn R Haverkort and Armin Zimmermann. 2017. Smart industry: How ICT will change the game! *IEEE internet computing* 21, 1 (2017), 8–10. https://doi.org/10.1109/MIC.2017.22

[5] Jay Lee, Hung-An Kao, Shanhu Yang, et al. 2014. Service innovation and smart analytics for industry 4.0 and big data environment. *Procedia Cirp* 16, 1 (2014), 3–8. https://doi.org/10.1016/j.procir.2014.02.001

[6] Biswajit Sarkar. 2016. Supply chain coordination with variable backorder, inspections, and discount policy for fixed lifetime products. *Mathematical Problems in Engineering* 2016 (2016). https://doi.org/10.1155/2016/6318737

[7] Edward Waltz, James Llinas, et al. 1990. *Multisensor data fusion*. Vol. 685. Artech house Boston, Norwood, United States.

[32] $https://support.office.com/en-us/article/workday-function-f764a5b7-05fc-4494-9486-60d494efbf33$

[33] https://pypi.org/project/workdays

[34] https://pypi.org/project/BusinessHours

A TESTING EXISTING METHODS

PintPy

How does it work? Pintpy appears to be a very powerful, rich library. It can verify whether the intended output unit actually represents the same measurement as the output unit (temperature unit in means the output unit also needs represent a temperature).

```
import pint
ureg = pint.UnitRegistry()

# PintPy Input:
(2 * ureg.meter + 2 * ureg.ft)
# Output:
<Quantity(2.6095999999999999, 'meter')>
```

PintPy is mainly designed to sum two unit systems and immediately convert them to one unit. The library can be tricked by summing a '0' amount of the desired output unit system to the actual input unit amount.

```
# Fool PintPy with this input:
(0 * ureg.meter + 2 * ureg.ft)
# Output:
<Quantity(0.6095999999999999, 'meter')>
```

Did any error occur? Despite it's vast set of unit systems, multiple errors occurred. The library misinterpreted some units, resulting in confusing error messages. Furthermore the system requires you to specify the units inside the code, which requires programmers to understand which units are being used.

Solutions of error prevention and consequences. This is where this library really shows it's negative side. The syntax requires hardcode programming the unit of a variable (such as a column in a tabulation). This means that we cannot dynamically change the unit, thus it is not very suitable unless we expect our personnel to all understand programming, and are comfortable with everybody being able to change the code!

A solution to this could be to write special functions that retrieve the correct attributes for PintPy or add dictionary values to supply the correct corresponding attributes into PintPy. This is certainly feasible but would result in multiple translations, as we first have to translate our input unit to a unit that is understood by the package, and vice versa. It is definitely a possibility and is not to be ruled out from future work, however preference was given to write an alternative solution as PintPy does not suit our 'freedom to choose input and output conversions easily' desire.

Conclusion. The programming library requires the programmer to understand which units he is converting, as the syntax demands unit coding. An alternative would be to integrate multiple functions to translate units to the correct unit for the PintPy program, or retrieve the correct attributes. As these alternative solutions would not yield a clean code experience, I consider it the main issue with this (otherwise) suitable package.

PiPy: Unit-convert

PiPy is a very simple to understand library which seems to imitate what PintPy does. It can combine two items and converts them into another unit, so it can take multiple units at the same time. This is a promising library as this potentially allows us to do complex conversions at the same time.

How does it work? The syntax is similar to PintPy, yet slightly more natural to interpret, as the desired output is on the last part of the code line.

```
from unit_convert import UnitConvert
# yards and kilometers are inputs, converted to miles
UnitConvert(yards=136.23, kilometres=60).miles
# Output
37.3597678005
```

Did any error occur? Yes, my first test of the temperature variable was not recognized as a measurement type. This surpsrised me a lot and I discovered that this library actually only converts data (computerstorage), time, distance and mass.

Solutions of error prevention and consequences. This would require adjusting the library itself, which in essence does not provide a 'off-the-shelf' solution. Furthermore it suffers from the same problem as PintPy, where it requires the programmer to hardcode the desired units inside the code (loosing versatility).

Conclusion. This is a limited library and not suitable for any complex implementation.

PiPy: Unit-converter

How does it work? Unit-converter allows us to specify exactly the specific scientific notation of units. This is really suitable to be applied in scientific situations, where often data is accompanied by the scientific notation.

Did any error occur? Yes, this library actually exposed many issues with unit notation. As the example code shows, the library expects temperatures to be accompanied by a special character: °C for Celsius. Unfortunately the program does not accept any other notation for temperature scales (Fahrenheit also succumbs this annotation requirement).

```
from unit_converter.converter import convert, converts
# The special character ° is required
converts('52°C', '°F')
# Output, but which unit?
'125.6'
```

Solutions of error prevention and consequences. The issue with the character requirement originated from the csv file exported from Excel. When using the standard CSV format in Excel, it is not encoded in 'UTF-8', required in order to add this special character. After implementing this additional character, errors occurred in other parts of the code but this could be overcome. The question becomes how versatile this software is in order to use it in multiple solutions.

Conclusion. This library is not useful for mass implementation due to the requirements for special characters, which can suddenly lead to errors.

String format: Additionally I would like to highlight that the library requires parameters in the string format. String formats are very versatile as they can represent any type of value and thus also are heavy data containers from a memory perspective. The aim for this research is not about memory space or computational speed but as this case is so excessive it is worth pointing out.

B CODE OF THE PROPOSED METHOD

I will clean up this section for the final hand-in.

In general:

- Import conversion csv for dictionary
- Option to create hardcoded conversion dictionary
- Convert units using dictionaries
- Import database csv and shape dataframe
- Cleanup all imported values
- Convert values

Import conversion csv for dictionary

```
import csv # Read csv to Dictionary
reader = csv.reader(open('unit_conversions.csv', 'r'), delimiter=';')
dict_unit_csv = {} # start with empty dictionary
for k, v in reader:
    dict_unit_csv[k] = v # add key and value to dict
```

Option to create hardcoded conversion dictionary

```
# Self-made hardcoded dictionary
dict_unit_hardcoded = {
    # Distance to Meter (SI) using * multiplication to go to SI
    'distance_SI_km':0.001, # kilometer
    'distance_SI_m':1, # meter
    'distance_SI_cm':100, # centimeter
    'distance_SI_mm':1000, # millimeter
    'distance_USCS_mi.':1609.344, # miles
    'distance_USCS_ft':0.3048, # feet
    'distance_USCS_in':0.0254, # inch
    # Volume to Liter (SI) using * multiplication to go to SI
    'volume_USCS_cu_in':0.016387064, # cubic inch
    'volume_USCS_cu_ft':28.316846592, # cubic feet
    'volume_USCS_cu_yd':764.554857984, # cubic yard
    'volume_USCS_bbl':158.987294928, # oil barrel
    'volume_SI_L':1, # Liter
    # Temperatures, note the lambda function
    'temperatures_USCS_°F': lambda x : ((5/9) * (x - 32)), # Fahrenheit to C
    'temperatures_USCS_F': lambda x : ((5/9) * (x - 32)), # Fahrenheit to C
    'temperatures_SI_°K': lambda x : (x - 273.15), # Kelvin
    'temperatures_SI_K': lambda x : (x - 273.15), # Kelvin
    'temperatures_SI_°C': 1, # Celsius
    'temperatures_SI_C': 1, # Celsius
    # Weights
    'mass_USCS_lb': 0.45359237, # Pounds
    'mass_SI_kg': 1, # Kilogram
    'mass_SI_g': 1000, # grams
}
```

Convert units using dictionaries

```
# Make universal dictionary reader for unit conversions

def convert_units_from_dict(dict_to_use, unit_subject, unit_system, unit_specs, data_in):
    '''
    Retrieves conversion units from dictionary to be multiplied\n
    Apply a function in case it is a function\n
    dict_to_use = dictionary used to retrieve value\n
    unit_subject = describes the subject of unit, e.g. temperature, length\n
    unit_system = describes the unit system USCS, Imperial, SI etc\n
    unit_specs = Specifies the exact unit used
    '''
    retrieval_value = unit_subject + '_' + unit_system + '_' + unit_specs
    if retrieval_value not in dict_to_use:
        # This will stop the program!
        raise RuntimeError("Unit "  + retrieval_value + " not found in dictionary. Please update data or dictionary.")

    x = dict_to_use[retrieval_value]
    if callable(x):
        return x(data_in) # applies the function as stated in dictionary
    else:
        return data_in * x # conversion is not a number, e.g. a function
```

Import database csv and shape dataframe

```
# Code comes here, needs cleaning up
```

Cleanup all imported values

```
def cleanup_data_values_return_float(data_in):

    message_error_string = " is the datatype value in database, but it must be a floating point or integer"

    if type(data_in) is bool:
        raise RuntimeError("Boolean " + message_error_string)

    if type(data_in) is str:
        data_in = data_in.replace(',','.')
        if data_in.count('.') > 1:
            data_in = data_in.replace('.','', data_in.count('.') - 1)

    try:
        data_in = float(data_in)
```

```
    except:
        raise RuntimeError("String " + message_error_string)

    return(float(data_in))
```

Convert values

```
# Code comes here, needs cleaning up
```