Data Wrangling in python: Walkthrough with a practical example

# Data Wrangling definition

Wrangling is just another fancy word for data preparation. It is the processing of data to make further exploration and analysis straightforward and less error prone. Sometimes it is as simple as opening an Excel formatted file in MS Excel, and you are well on your way. Most times, as we will surely experience, it is not that simple. Most data in the world are inherently messy or ‘untidy’, if you will. Sometimes, the data is arranged in rows, but you want them in the form of columns to make the next step of your analysis possible. Most times, this is just one step of the metaphorical staircase that is data cleaning.

Here we will follow a typical data wrangling framework for cleaning some part of a sample dataset. This is a real-world dataset, as you will find out soon, that will help us in getting our hands dirty with the first step of the complete data analysis process.

You can download the complete code base and the datasets used in this example from the github repo using this link: <https://github.com/rashokanand/wrangling-armenian-job-postings-kaggle>

Here you will find the Jupyter notebook along with the zip folder used in the analysis of this dataset.

## Prerequisite skills

Input and output (IO) with Python, working with Jupyter notebooks (optional. Although I will be referring to code segments in cells, you may just as well type up a .py script)

Introductory skill with pandas.

# Background of dataset used

The dataset that we are using can be found on Kaggle. It contains over 19000 job posting in the period between 2004 to 2015. The data was mined using text mining techniques from an Armenian HR portal. Though most of the heavy lifting of data wrangling is already been done, it still needs a little work to make it useable. It should be noted that our objective is not to completely and exhaustively clean the dataset, but to get a first hand taste of what data cleaning is and how it can be done in the real world. The objective is to use a consistent and reproducible framework deliberately designed to aid in systematizing data wrangling.

# Framework for data wrangling

Wrangling literally means ‘to round up, herd or take charge of (livestock)’. At least that’s what the Oxford dictionary definition says. It is probably safe to say that, it’s true in the case of data wrangling as well. Here we need to get the data in order to make the next steps of data analysis possible.

We here will follow a three step process to complete the data wrangling process: Gather, Assess and Clean.

# Gather

This consists of getting the data into the programming environment or the spreadsheet application that we are using presently. This may include the actual data download, extract and then import. For the purpose of this article, we will download the files manually, then proceed doing the rest of the process manually.

Using the link in the references, go ahead and download the zip file. We may need to register an account in Kaggle for this. The downloaded file’s name is ‘armenian-online-job-postings.zip’. Do not unzip it just yet. We will do it directly using python in a jupyter notebook.

Create a folder which will be your current working directory. Move the zip into this folder. Start up the Jupyter notebook in this directory. You may need to change your directory in the terminal first using ‘cd’. Then through the command ‘Jupyter notebook’ launch the notebook in your browser. If you do not understand what just happened, you need to first understand what are jupyter notebooks and how to install and run them. The best resource for this purpose (that I found to be most helpful) is jupyter.org. It contains tutorials and examples for installing and running the same successfully. Comment if you are stuck anywhere in this process.

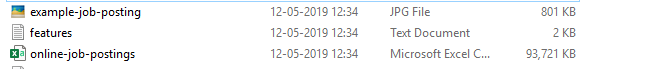
Alright then, let’s go ahead. Using the zipfile module to extract the files in the archive is simple using the ‘extractall’ method

# Extract all contents from zip file

with zipfile.ZipFile('armenian-online-job-postings.zip', 'r') as myzip:

myzip.extractall()

This code works exactly like how when we right click on an archive and click on ‘extract all files here’. Reading up on the documentation of zipfile, or for that matter any module that you would like help on is a very valuable and underrated skill.



After the files are extracted we are now looking at a collection of 3 files. Our interest is with the ‘online-job-postings.csv’ file.

Using pandas we can read in the dataset like so.

# df = pd.read\_csv('online-job-postings.csv')

The above code will work only when the csv is in the same directory as your jupyter notebook. Or we need to pass the complete path as the argument to the read\_csv method of pandas. It then stores the output dataframe into the variable ‘df’.

Congratulations, you are done with the ‘Gather’ step of the data wrangling process. Two more to go.

# Assess

After gathering the data, next comes assessing for inconsistencies in it. The assess step can be commonly confused with exploratory data analysis. It is not though. Exploratory data analysis (EDA) deals with descriptive statistics and finding patterns within the data. It is to understand the data before actually coding up regression or classification models upon it. Assessing in Data wrangling deals with identifying issues that may prove to be obstacles for data analysis or sometimes even break it. Common issues include text (strings) in a numeric column. Inconsistent data values such as having weight in both kilograms as well as pounds in the same weight column. There may even be errors in the dataset that need to be replaced with some common value, for example ‘N/A’. This is a common error we face when we work with excel files.

During the Assess step, the pandas’s .info(), .value\_counts(), .head(), and .tail() methods are your best friends. If you do not know them, I would suggest you read up the official docs for pandas and at least for now get familiar with them. Programmatic assessment is way easier with these tools in your data cleaning toolkit. If it is a small dataset, opening it in excel and cleaning it manually is also simple. However, if your dataset spans thousands of rows (19k in this case), it is definitely rewarding to use these skills. Programmatically assessing and cleaning a dataset makes the whole process reproducible for anyone, or for your future self, looking back at your past projects.

The first issue that we will deal with is in the ‘StartDate’ Column of the table. Using the value\_counts() method on the StartDate series of the dataframe we can see all the different values in that column ordered by the frequency of occurrence of that value. We can see that ‘ASAP’ is mentioned more than any other value. Exactly 4754 times. But if you see the output of the value\_counts() function, you will see that a little less frequently but nonetheless there are other values that essentially mean the same. Values such as upon hiring, as soon as possible, immediate, immediately, etc all mean the same as ASAP (at least in spirit). Now if we do not correct for these, then we are not painting the complete picture. We would end up drastically underestimating the number of job posts that required immediate joining. Let us note down this issue so we can fix it during the next step. It is important to note that, in the Assess step, our one and only job is to scan through for errors and inconsistencies that can hamper or break our data analysis. We do not start cleaning and fixing just yet. This comes later. If we do this haphazardly, then our efficiency suffers a hit.

Hence, moving on…. In this dataset, we find that the column names are not so descriptive. The names of the variables are easier to remember during analysis if they have descriptive names such as AboutCompany instead of AboutC. This is more of a convention issue rather than an actual data issue, critical, nonetheless. Having easy to remember and descriptive variable names would save us a chunk of time spent recalling them, that could have been spent on more pressing and productive tasks.

For now we will concentrate on only these two issues. Instead of striving to completely clean the complete dataset for all errors imaginable, it is easier and more efficient to clean only those errors that would immediately impact your further analysis. This systematic data wrangling process is completely iterable. As and when we feel that there are more issues with the data that need to be corrected, we can come back and add the necessary steps. And since we are completely documenting our every step, we will never be lost.

# Clean

Now that we have identified the issues that we need to correct, cleaning is fairly straightforward. We will follow a three step process for cleaning each issue: Define, Code and Test. Since we have identified 2 issues, there will be 2 sets of Define, Code and Test steps.

Here we will not be going into the specifics of how to clean the identified issues. You have the jupyter notebook hosted in github for that. The objective is to get familiar with the process of systematically structuring our cleaning process. Nothing more!

## Define

In this step, the issue is broken down into pseudo code. Pseudo code is a set of instructions in our tongue (English instructions) that if converted to programming statements using syntax of the relevant programming language, runs them. Refer to the jupyter notebook. It contains the breakdown of how to fix each issue in pseudocode before the actual code. The importance of defining each step properly before sitting down and coding it up, is that we will have the picture in mind. What are we doing and why, and then what would it achieve? It is easier to back track and understanding what we were thinking when we are referring to our old projects, if they followed these conventions.

## Code

Here we convert our pseudocode that we have from the previous step to code. It is important to store the cleaned data into a new variable, so that the original data remains unaffected. So if we screw up big time, we have the old dataframe to fall back on. Here I stored the data into df\_clean, before doing any of the cleaning tasks.

## Test

Now that we have cleaned and stored the clean data into a new variable, we may test it to see if we actually accomplished what we defined in the define step. You may find that the .info() and .value\_counts() methods that we so dearly used in the Assess steps are again useful in this section. After we confirm that the issue we tackled has indeed been rectified, we can move on to tackling the next issue. We need to iterate over these three steps over until there are no more issues left to tackle.

# References

Link to dataset: <https://www.kaggle.com/udacity/armenian-online-job-postings>

<https://www.jstatsoft.org/article/view/v059i10> Tidy data by Hadley Wickham

GitHub repo: <https://github.com/rashokanand/wrangling-armenian-job-postings-kaggle>