Heidi Beezub

DATA 520 Introduction to Python

Project Report : The DCM 9522

December 6, 2017

**Background**

Part of my job as a “Managerial Cost Accountant Program Analyst” for the Erie VA Medical Center, I have to correct rejected data.

***What is a reject?*** A reject is data that the system collects but then cannot process or put into the correct category. Rejects are typically a ‘new’ item that the simply needs the interface created or built in the system. Sometimes this is an existing item that is either coming into the system in a new way.

***The interface:*** matches the data into the correct department. The system matches the product information (feeder key and feeder system).

My project takes an unwieldy 6,000+ report & makes it an excel file which can easily be sorted & searched.

**Process/Methodology**

***Source data***

My first step was to take a VERY critical look at the structure of the report:

-The beginning of the report contains multiple report & data descriptor & disclaimer lines.

-Each page has a header which includes the following lines:

report name line “DCM 9522”

run date line

bill of materials line

Fiscal year line

Dept line (the only data I need in the header)

Blank line

Data header lie 1

Data header line 2

Data header line 3 (---- dash line)

-After the header is the data. In addition to the product information that I want (prod description, IP number, sequence number, feeder system and feeder key) it includes a vast quantity of ‘bill of material’ (BOM) data I don’t need to process rejects. A product can have multiple feeder systems & multiple feeder keys.

-The header can bisect the data at any point.

-The end of each BOM ends with a set of equal signs ‘=’ the length varies depending on the cost of the items.

-The bottom of each page is a ‘government use only’ line

-The end of the report is an ‘end of report sequence of asterisks.

***The desired end result***

A csv (excel file) that I will be able to filter to check items, sort to identify duplicate products and otherwise manipulate the data.

***The code***

*(Pseudocode)* First, I wrote down (pen & paper) what I needed to do.

I decided to work in small chunks starting with the easiest part first (the IP number, sequence number, feeder system & feeder key), these are specifically identified and easy to extract.

I truncated my source data from 6,600 pages to 37 pages to write my preliminary code.

I started with the code to extract the IP number. One at a time, I added code for the sequence number, feeder system, feeder key and Department.

I started using a module method to extract the various items, however, when I got to the point of extracting the product description I realized there was no way to ‘keep’ variable values between the functions. I needed to restart in a different direction using one module of code to keep track of variables for when to extract the product description.

***Code to extract product description***

Unlike the other items, the product description has no key words at the beginning of the line. Since there is a ‘=’ line at the end of each BOM item I can use this to set my counter. However,

the product description was more challenging. There were no key words at the beginning of line. I would have to keep track of a prior line to know when the next line would be the product description. This is where familiarity with the data was key. At the very END of each Bill of materials was an “=” line. For the first item description I would process after the first header.

I used 'counters' to keep track of when I came to the end of one item (the ‘=’ line), and to keep track of when the first item description was processed.

**Testing**

Once the code was all written (and working) I expanded my source data slowly adding additional pages 127, 500, 1000, 2,000 & 3,000 pages before finally running the entire 6000 pages.

I was surprised to find that my debugging continued each time I added more source data.

As I expanded my source data, I would sometimes truncate the test data to a few pages with challenging cases. During testing I reviewed the csv file data to insure the formatting & data was accurate. This involved not only ‘eyeballing’ the data but also section comparison between the 9522 and csv file to insure accuracy.

**Lessons Learned**

***Small, Small, Small*** Even though I had in my mind to start small I still started with too much code. As part of my design I need to break down my steps as well.

***The Notebook*** I wish I had kept a notebook (with notes from beginning to end.) I also which I would have kept track of my time (so I could compare programming vs testing/verifying time).

***My own genius*** I am proud of myself for the way I handled finding the first line after the header (without getting any other line after the header). By defining the '---' (dash line) through a variable, I could then change the variable so it would never find the dash line again.

***#notes*** I forced myself to add many comments. I know it will be at least 2 weeks before I go to my next step of configuring the 9522 for another station (hospital).

***The end game*** The code works for my hospital’s data. Once I know the code works for other sites data I can present it to our region for use (region 4 of 11 regions). After feedback from my region I would then hope to present it nationally.

**Codes Sources:**

None

I did google some code help for how to get rid of commas (replace not strip)

<https://www.safaribooksonline.com/library/view/python-cookbook-3rd/9781449357337/ch02s11.html>