Data Cleansing and Integration Project

Scott Lee

Task 1 Auditing and Cleansing the Job Dataset

The first task was to load the data into a pandas dataframe to visually inspect the data and begin exploratory analysis. Each column of the dataframe was inspected individually and then compared to the desired format.

Id: [Integer] 8 digit Id of the job advertisement

Data converted to a list of strings for analysis to ensure everything was 8 digits. Then converted back to integer and levet as is.

Title:[String] Title of the advertised job position.

Regular expressions used to filter out some text of no substantive value. Lambda function was then used to populate a new dataframe and analysis completed.

Final “Location” values are all uppercase strings. Some of the main issues with the data were duplicates due to different case types, excessive special characters that did not provide real semantic meaning. The code was adapted to not filter out “/” and “+” as they had semantic meaning e.g. “C/C++ Developer” relied on it to show the two programming languages required by a developer and the “+” is descriptive of the language.

Additionally, some of the “Title” data also contained “Location” data. In this instance, using a for loop construct the location data was filtered out of the title data if it was a match to the corresponding location data.

Sequence matcher and counter were used to eliminate miss spellings and typo versions of the same job title.

Location: [String] Location of the advertised job position

Regex expression (clean_location function) used to clean the location data incorporated within a function. All were converted to uppercase for final presentation and to remove duplicates.

Company: [String] Company (employer) of the advertised job position

Regular expressions were incorporated into the function similar to location. Final output were all of type string.

As was shown in the Module 4 tutorials, the company names were prone to errors for the same name being entered with different spellings. The SequenceMatcher module from difflib ws sussed in conjunction with Counter from the collection modules.

I then incorporated the 'match_highfreq_To_lowfreq' and 'similar' functions from the tutorials to reconcile some of the differences by identifying the company names that appeared at a low frequency to those they closely match at a higher frequency.

Prior to using SequenceMatcher there were 8,671 unique company names, however after using it there were 8,622 names, a reduction of 49 duplicates.

ContractType: [String] The contract type of the advertised job position, it could be full_time, part_time or non_specified.

After analysis there were only four unique values:

- No value
- 'full_time'
- '-'
- 'part_time'

A for loop construct with a nested set of if statements were used to replace the unknown values with 'non-specified'.

Unique values were then tested again after the above programming sequence had run confirming the only three data entries within the 'ContractType" column were:

- 'non_specified'
- 'full_time'
- 'part_time'

ContractTime: [String] The contract time of the advertised job position, it could be permanent, contract or non-specified

Five unique values existed within the 'ContractTime' column:

- 'permanent'
- '-'
- 'contract'
- nan
- ' '

A similar construct to 'ContractType' was employed.

Unique values were then tested again after programming sequence had run confirming the final unique values were as desired:

- ‘permanent’
- ‘non-specified’
- ‘contract’

Category: [String] The Category of the advertised job position, e.g., IT jobs, Engineering Jobs, etc

Analysis did not find much wrangling requirements for the ‘Category’ values. There were only 8 unique values to begin with, all of which appear to be valid. Data type as string was confirmed as string.

Salary: [Float] Annual Salary of the advertised job position, e.g., 80000

Some of the inconsistencies in the ‘Salary’ data set included the addition of ‘per annum’ to a field and the use of ‘k’ to represent ‘000’. There were also ‘-’ and nan values and ranges. These inconsistencies were all dealt with using regular expressions embedded in the function ‘Clean Salary’.

As the desired output was a float, any nan or ‘-’ values were converted to a ‘0’ value.

Final output for this column was of type float.

OpenDate: [String] The opening time for applying for the advertised job position, e.g., 20120104T150000, means 3pm, 4th January 2012.

‘OpenDate’ and ‘CloseDate’ were treated in a similar fashion. First a regular expression was used to check for inconsistencies. For example sometimes a value greater than 12 would appear in the place of the month digits, indicating that the month and day were the wrong way around. These inconsistencies were then manipulated to produce a valid value.

The ‘datetime’ module was then incorporated to convert the string into correct date format, clearly displaying the date and time.

CloseDate: [String] The closing time for applying for the advertised job position, e.g., 20120104T150000, means 3pm, 4th January 2012.

‘OpenDate’ and ‘CloseDate’ were treated in a similar fashion. First a regular expression was used to check for inconsistencies. For example sometimes a value greater than 12 would appear in the place of the month digits, indicating that the month and day were the wrong way around.

The ‘datetime’ module was then incorporated to convert the string into correct date format, clearly displaying the date and time.

SourceName: [String] The website where the job position is advertised.

This dataset was checked for being the correct data type of string. Everything was converted to uppercase to remove duplicates.

Final Formatting of the DataFrame

The dataframe ‘df’ was then reformatted to ensure all the columns appeared in the correct order and were consistent with the desired output in the assignment brief. ‘df’ was then exported to a csv in preparation for part 2 of the assignment.

Task 2: Integrating the Job Datasets

The ‘dataset2.csv’ data set was loaded in and inspected for initial inconsistencies. Some immediate inconsistencies that were observed included:

- Discrepancies in column naming conventions
  - Opening vs OpenDate
  - Closing vs CloseDate
  - Job Title vs Title
  - Organisation vs Company
  - Salary per month vs Salary (different period version of the same thing)
- Contract Type actually referred to the same data as ContractTime in ‘dataset1’.

The ‘dataset2.csv’ was loaded in as ‘df2’. The dataframe from part 1 referred to as ‘df1’ was used as the desired format so adjustments were made of the new dataframe to resolves schema issues.

The column titles of the new dataframe (‘df2’) were amended to match that of the first dataframe ‘df1’ when they had the same meaning.

Also data type was checked for consistency.

An ‘Id’ column was added to support unique naming of each individual entry, thus creating a primary key for the dataset.
The ‘Salary per month’ column in ‘df2’ was recognised as a fraction of the ‘Salary’ column in ‘df1’. ‘Salary per month’ was multiplied by 12 and renamed as ‘Salary’ so that it could be joined with ‘Salary’ in ‘df1’.

The ‘concat’ method was used to concatenate the different data frames. This was appropriate as after I had completed my manipulation of ‘df2’ I had matching columns to join on, including “Id” which I used as a primary key to uniquely identify.

I assumed that no other combination of attributes would have guaranteed a unique result (hence I used an ‘Id’ field). For example it is entirely possible that the same company could post for multiples of the same position, causing 2 or more job postings to have identical attributes in every way but still separate listings.