DataEng S22: Data Transformation In-Class Assignment

Submit: Make a copy of this document and use it to record your results. Store a PDF copy of the document in your git repository along with any needed code. Submit the in-class activity submission form by Friday at 10:00 pm.

Initial Discussion Questions

Discuss the following questions among your working group members at the beginning of the week and place your own response (or that of your group members) into this space. Be sure to mark each response as either your own or that of the group (both types of responses are welcome)

In the lecture we mentioned the benefits of Data Transformation, but can you think of any problems that might arise with Data Transformation?

- ***The cost could be expansive***
- ***Lack of expertise could result some problems***

Do you think data transformation or validation should come first in your pipeline? Why or why not?

- ***Validation should come first because we might need to use transformation twice if we used transformation first.***

ETL (Extract, Transform, Load) is a common pipeline process. Describe the delineation of each of these separate activities. For example, how is extraction different from transformation?

- ***Extract data from Operational DBs***
- ***Transform to a Star Schema or Snowflake Schema***
- ***Load the data into the Data Warehouse***

Source: Data Transformation slides

Pandas Review

Recall in Week 2: Data Gathering, recall that we introduced several table-level pandas methods:

- `df.drop()`
- `df.dropna(axis=0, how='any')`
- `df.rename(columns=newColumns)`
- `df.concat([df1, df2])`

Additionally during week 2, we performed data cleaning at the cell level, by accessing cell data as strings with the df[0].str attribute. This attribute allows us to apply standard python string operations on the cells of the dataframe, like .split() .contains() .replace() .capitalize() etc.

These are actually all types of data transformations! Each of these methods can be used to clean, fix, and repackage the data as needed.

Explore the activity below to learn about more complex transformations. To encourage you to explore the pandas methods available, we have not told you which methods to use in all the cases. Use the examples above to consider what other methods may exist. For example we already know about drop, does insert exist? How can you find more information about a method's arguments or examples of usage?

Keep in mind these handy methods which give you more information about your dataframe:

- `df.head()`
- `df.tail()`
- `df.info()`
- `df.shape`
- `df.describe(include=[np.number])`
- `type(df)`
- `type(df['columnName'])`

If you want more general practice with pandas, consider running through the W3 tutorial linked below. It includes examples, documentation, exercises, and quizzes to help you feel comfortable working with pandas: https://www.w3schools.com/python/pandas/default.asp

A. Filtering

We'll be using this book dataset from the British Library: link

Use python and pandas to filter this data by dropping these columns: **`Edition Statement, Corporate Author, Corporate Contributors, Former owner, Engraver, Issuance type, Shelfmarks`**

Do this two ways. First use the DataFrame **`drop()`** method. Then do the same with the usecols argument of pandas.read_csv()

Hint: are you dropping rows or columns? Is there an argument for that in the drop method? What types of values does the usecols argument expect?

B. Tidying Up the Data

In the book data, notice that the “Date of Publication” column has many inconsistencies. Update all of the data in this column to be consistent four digit year values. Specifically,

- Remove the extra dates in square brackets, wherever present: e.g., 1879 [1878] should be converted to 1879. **Done**
- Convert date ranges to their “start date”: e.g., 1860-63; 1839, 38-54 **Done**
- Remove uncertain dates and replace them with NumPy’s NaN: [1897?] **Done**
- Convert the string nan to NumPy’s NaN value **Done**
- Finally, update the type of the “Date of Publication” column to be numeric (not string, not object) **Done**

The “Place of Publication” column of this data set is also untidy. Transform all of the values in this column to be only the name of the city. If the city name is not found in the string, then the name of the country. If neither are present then transform to the string “unknown”.

C. Tidying with applymap()

See this list of USA towns that have universities: uniplaces.txt. This data was originally created for another purpose and contains artifacts of that. For example it is alphabetized by the name of the state where the university is located. The state is listed once and then the universities present in that state are listed on the lines below. Additionally there are extra punctuation marks to designate separation of the town and the university name (), and an artifact number at the end [2]. We would like to change the data to have 3 columns containing the state, city, and university.

Task:

Use the applymap() method to apply a custom function to the data. This should transform it into a tidy list of city, town, and university. Details below.

Task Details

There are a lot of examples you can find for how to use applymap(). This example from Geeks4geeks uses a lambda function. A lambda function in python is a simple function that can be accomplished in one line. The method applymap() then applies that function to each row of the dataframe. Therefore df.applymap(lambda x: len(str(x))) will operate on the dataframe called

df. For each element of each row of df, it applies the lambda function, naming the element x. This lambda function finds the length of x.

In python you can pass functions as arguments to another function. In this example below, you can choose to emphasize your text by either shouting it or whispering it, depending on which function you pass to emphasize().

```
>>> def shout(text):
...         return text.upper()
...
>>> def whisper(text):
...         return text.lower()
...
>>> def emphasize(myfunc, s):
...         if s[0:5] == 'Hello':
...                 return myfunc(s[0:5]) + s[5:]
...         return s
...
>>> emphasize(shout, "Hello World!")
'HELLO World!'
>>> emphasize(whisper, "Hello World!")
'hello World!'
```

In our applymap() example above, we applied a lambda function to each row of the dataframe. Instead, you can apply your own custom function.

The method applymap() takes a function as input and applies it to the dataframe it is called on. Write a custom function which handles the uniplaces.txt data and reformats it as 3 columns for state, city, university.

Hint: first create a dataframe from this data, with the desired columns. Then use applymap to clean out the extra artifacts

D. Decoding

Similar to C-Tran, TriMet also produces breadcrumb data for its buses. Here is a sample for one bus on one day of October 2021: link to breadcrumb data

One column of the TriMet breadcrumb data is called “OCCURRENCES”. Our contact at TriMet explained this field as follows:

OCCURRENCES – number of times a point appeared in the dataset. This is to clean up some of the data because sometimes when the vehicle is stationary it will replicate multiple instances at the same point. This consolidates those into a single record.

This encoding of multiple breadcrumbs into a single record helps to save space, but for analysis we typically need to decode it so that all of the records can be analyzed. Often decoding consists of exploding one row out into multiple rows.

Your job is to decode records with OCCURRENCES > 1 into replicated records in a DataFrame. So for example, a sequence of records like this:

```
4313660399,03411,B,29OCT2021:08:36:17,29OCT2021:00:00:00,30977,-
122.844715,45.503493,0,223428.48,8,12,0.7,1,Y,TRANS,31OCT2021:06:06:40
4313660400,03411,B,29OCT2021:08:36:22,29OCT2021:00:00:00,30982,-
122.8448,45.503335,32,223487.54,8,11,0.7,9,Y,TRANS,31OCT2021:06:06:40
4313660401,03411,B,29OCT2021:08:36:57,29OCT2021:00:00:00,31017,-
122.844858,45.503212,5,223533.47,10,10,1.3,2,Y,TRANS,31OCT2021:06:06:40
```

Should be expanded to a sequence of records like this:

```
4313660399,03411,B,29OCT2021:08:36:17,29OCT2021:00:00:00,30977,-
122.844715,45.503493,0,223428.48,8,12,0.7,1,Y,TRANS,31OCT2021:06:06:40
4313660400,03411,B,29OCT2021:08:36:22,29OCT2021:00:00:00,30982,-
122.8448,45.503335,32,223487.54,8,11,0.7,9,Y,TRANS,31OCT2021:06:06:40
4313660400,03411,B,29OCT2021:08:36:22,29OCT2021:00:00:00,30987,-
122.8448,45.503335,32,223487.54,8,11,0.7,9,Y,TRANS,31OCT2021:06:06:40
4313660400,03411,B,29OCT2021:08:36:22,29OCT2021:00:00:00,30992,-
122.8448,45.503335,32,223487.54,8,11,0.7,9,Y,TRANS,31OCT2021:06:06:40
4313660400,03411,B,29OCT2021:08:36:22,29OCT2021:00:00:00,30997,-
122.8448,45.503335,32,223487.54,8,11,0.7,9,Y,TRANS,31OCT2021:06:06:40
4313660400,03411,B,29OCT2021:08:36:22,29OCT2021:00:00:00,31002,-
122.8448,45.503335,32,223487.54,8,11,0.7,9,Y,TRANS,31OCT2021:06:06:40
4313660400,03411,B,29OCT2021:08:36:22,29OCT2021:00:00:00,30907,-
122.8448,45.503335,32,223487.54,8,11,0.7,9,Y,TRANS,31OCT2021:06:06:40
4313660400,03411,B,29OCT2021:08:36:22,29OCT2021:00:00:00,30912,-
122.8448,45.503335,32,223487.54,8,11,0.7,9,Y,TRANS,31OCT2021:06:06:40
4313660401,03411,B,29OCT2021:08:36:57,29OCT2021:00:00:00,31017,-
122.844858,45.503212,5,223533.47,10,10,1.3,2,Y,TRANS,31OCT2021:06:06:40
4313660401,03411,B,29OCT2021:08:36:57,29OCT2021:00:00:00,31022,-
122.844858,45.503212,5,223533.47,10,10,1.3,2,Y,TRANS,31OCT2021:06:06:40
```

This is because the second breadcrumb in the example (4313660400) has an OCCURRENCES value of 9. Note that for this exercise it is OK to duplicate the 3VEH13660400

After you have expanded out the multiple rows, be sure to clean up the dataframe if necessary. It should have the same number of columns that you started with, in the same order, and they should all be named the same as when we started.

Hint: How can you decode a row into multiple rows in pandas? While it may be tempting to try to iterate through the dataframe and append new rows, instead consider table-level pandas methods that you can use. If any DataFrame methods you want to use are not available on a Series, is there an equivalent method for the Series?

E. Filling

The TriMet data, linked above, is missing some values in the VALID_FLAG column. Use the **`pandas.DataFrame.ffill()`** method to fill in the missing data.

Hint: How can you check for bad data like NaN or duplicates in a DataFrame? How can you find all the unique values in a column? For a column named like VALID_FLAG, what do you think are the expected values?

F. Interpolating

The TriMet breadcrumb data, linked above, is missing some values in the ARRIVE_TIME column. Use the **`pandas.DataFrame.interpolate()`** method to fill in the missing time data. The interpolate method fills in NAN values in a pandas DataFrame or Series. There are many different methods of interpolation that you can specify for different use cases. Be sure to use the 'linear' interpolation method which fills in the value based on previous values, ignoring the index, and equally spacing the missing values.

Hint: What is the frequency of the bus datapoints? Do we expect them every minute, every few seconds, etc? Does interpolate achieve this automatically? If not, how can you adjust it to do so?

Could you have used the **`interpolate()`** method for problem E above?

G. Melt

This activity will be filled in for Wednesday.

H. Merge

This activity will be filled in for Wednesday.

I. Transformation Visualizations

This activity will be filled in for Wednesday.