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 5:00 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?

Tarun's Response => There won't be problem, unexpected duplicate, incorrect indexing, null values
My Response => increased cost in terms of time complexity, dependency, complexity of data

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

Tarun's Response =>validation should come first
My Response => Yes, Validation should come before and after transformation

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 first, transform data into 0s and 1s, load the data into pipeline

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()

Answer:

=> With the drop() method of DataFrame, I mentioned argument columns and mentioned the list of columns to drop.

CODE:

newBooksDF=booksDF.drop(columns=['Edition Statement', 'Corporate Author', 'Corporate Contributors', 'Former owner', 'Engraver', 'Issuance type', 'Shelfmarks'])

print(newBooksDF)

```
In [107]: newBooksDF=booksDF.drop(columns=['Edition Statement', 'Corporate Author', 'Corporate Contributors', 'Former owner', 'Engraver

          print(newBooksDF)
```

```
      Identifier      Place of Publication Date of Publication  \
0            206                    London         1879 [1878]
1            216  London; Virtue & Yorston               1868
2            218                    London               1869
3            472                    London               1851
4            480                    London               1857
...          ...                       ...                ...
8282     4158088                    London               1838
8283     4158128                     Derby          1831, 32
8284     4159563                    London          [1806]-22
8285     4159587      Newcastle upon Tyne               1834
8286     4160339                    London           1834-43

                   Publisher  \
0           S. Tinsley & Co.
1               Virtue & Co.
2      Bradbury, Evans & Co.
3             James Darling
4       Wertheim & Macintosh
```

=> WIth the usecols argument of pandas.read_csv() method, and added columns which I want to pick and did not mention the columns which I want to drop.

CODE:

df = pd.read_csv('./books.csv', usecols=["Identifier", "Place of Publication", "Date of Publication", "Publisher", "Title", "Author", "Contributors", "Flickr URL"])
print(df)

```
In [46]: df = pd.read_csv('./books.csv', usecols=["Identifier", "Place of Publication", "Date of Publication", "Publisher", "Title", "
         print(df)
```

```
      Identifier      Place of Publication Date of Publication  \
0            206                    London         1879 [1878]
1            216  London; Virtue & Yorston                1868
2            218                    London                1869
3            472                    London                1851
4            480                    London                1857
...          ...                       ...                 ...
8282     4158088                    London                1838
8283     4158128                     Derby            1831, 32
8284     4159563                    London           [1806]-22
8285     4159587       Newcastle upon Tyne                1834
8286     4160339                    London             1834-43

                    Publisher  \
0            S. Tinsley & Co.
1               Virtue & Co.
2       Bradbury, Evans & Co.
3              James Darling
4        Wertheim & Macintosh
...                       ...
8282                     NaN
8283          M. Mozley & Son
8284  T. Cadell and W. Davies
```

Hint: are you dropping rows or columns?
I am dropping columns.

Is there an argument for that in the drop method?
Yes, the argument is columns and it expects names of columns to drop. Or we can use labels and axis arguments to drop the column. Labels mention the string name s of columns to drop and axis 1 indicates to drop the columns not the index.
So labels, axis=1 is same as columns=labels.

What types of values does the usecols argument expect?
It expects list of columns and all elements must either an integer indices into the document columns or strings that correspond to column names

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

For this task, I used the str.extract() method. It uses a regular expression and returns the column or a dataframe which matches the regular expression. If suppose the value of one of the rows doesn't match it replaces it with NaN. So for all the cases mentioned above this regular expression works and returns the data which satisfies all the above conditions for “Date of Publication”. I have kept its expand argument as False because I wanted to access the return value as a series and replace the column in the original dataframe. And to update the type of the column “Date of Publication” I used the pandas to_numeric() method.

CODE:

df['Date of Publication'] = df['Date of Publication'].str.extract(r'^(\d{4})', expand=False)

print(df)

df['Date of Publication'] = pd.to_numeric(df['Date of Publication'])

```
In [47]: df['Date of Publication'] = df['Date of Publication'].str.extract(r'^(\d{4})', expand=False)
         print(df)

      Identifier      Place of Publication Date of Publication  \
0            206                    London                1879
1            216  London; Virtue & Yorston                1868
2            218                    London                1869
3            472                    London                1851
4            480                    London                1857
...          ...                       ...                 ...
8282     4158088                    London                1838
8283     4158128                     Derby                1831
8284     4159563                    London                 NaN
8285     4159587       Newcastle upon Tyne                1834
8286     4160339                    London                1834

                  Publisher  \
0           S. Tinsley & Co.
1               Virtue & Co.

In [48]: df['Date of Publication'].dtype

Out[48]: dtype('O')

In [49]: df['Date of Publication'] = pd.to_numeric(df['Date of Publication'])
         df['Date of Publication'].dtype

Out[49]: dtype('float64')
```

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”.

In [84]:
```
df['Place of Publication']=df['Place of Publication'].str.extract(r'([^\s]+)').replace(r'[^\w\s]+', '',regex=True)
print(df)
```

```
      Identifier Place of Publication Date of Publication  \
0            206               London         1879 [1878]
1            216               London                1868
2            218               London                1869
3            472               London                1851
4            480               London                1857
...          ...                  ...                 ...
8282     4158088               London                1838
8283     4158128                Derby           1831, 32
8284     4159563               London          [1806]-22
8285     4159587            Newcastle                1834
8286     4160339               London             1834-43

                    Publisher  \
0            S. Tinsley & Co.
1                Virtue & Co.
2       Bradbury, Evans & Co.
3              James Darling
4        Wertheim & Macintosh
```

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.

In [100]:
```
import re
uniplaces = []
with open('./uniplaces.txt') as file:
    for line in file:
        if '[edit]' in line:
            state = line
        else:
            #print(re.findall(r'\(.*?\)', line)[0])
            city=line.split(' ')[0]
            university=re.findall(r'\(.*?\)', line)
            uniplaces.append((state, city, university))
print(uniplaces)
```

```
nds)']), ('Kentucky[edit]\n', 'Wilmore', ['(Asbury University, Asbury Theological Seminary)']), ('Louisiana[edit]\n', 'Ba
ton', ['(Louisiana State University, Southern University)']), ('Louisiana[edit]\n', 'Grambling', ['(Grambling State Unive
rsity)']), ('Louisiana[edit]\n', 'Hammond', ['(Southeastern Louisiana University)']), ('Louisiana[edit]\n', 'Lafayette',
['(University of Louisiana at Lafayette)']), ('Louisiana[edit]\n', 'Monroe', ['(University of Louisiana at Monroe)']),
('Louisiana[edit]\n', 'Natchitoches', ['(Northwestern State University)']), ('Louisiana[edit]\n', 'Ruston', ['(Louisiana
Tech University)']), ('Louisiana[edit]\n', 'Thibodaux', ['(Nicholls State University)']), ('Maine[edit]\n', 'Augusta',
['(University of Maine at Augusta)']), ('Maine[edit]\n', 'Bar', ['(College of the Atlantic)']), ('Maine[edit]\n', 'Brunsw
ick', ['(Bowdoin College)']), ('Maine[edit]\n', 'Farmington', ['(University of Maine at Farmington)']), ('Maine[edit]\n',
'Fort', ['(University of Maine at Fort Kent)']), ('Maine[edit]\n', 'Gorham', ['(University of Southern Maine)']), ('Maine
[edit]\n', 'Lewiston,', ['(Bates College)']), ('Maine[edit]\n', 'Orono', ['(University of Maine)']), ('Maine[edit]\n', 'W
aterville', ['(Thomas College, Colby College)']), ('Maryland[edit]\n', 'Annapolis', ["(United States Naval Academy, St. J
ohn's College)"]), ('Maryland[edit]\n', 'Chestertown', ['(Washington College)']), ('Maryland[edit]\n', 'College', ['(Univ
ersity of Maryland, College Park)']), ('Maryland[edit]\n', 'Cumberland', ['(Allegany College of Maryland)']), ('Maryland
[edit]\n', 'Emmitsburg', ["(Mount St. Mary's University)"]), ('Maryland[edit]\n', 'Frostburg', ['(Frostburg State Univers
ity)']), ('Maryland[edit]\n', 'Princess', ['(University of Maryland Eastern Shore)']), ('Maryland[edit]\n', 'Towson',
```

```
In [101]:  uniDF = pd.DataFrame(uniplaces, columns=['State', 'City', 'University'])
           print(uniDF)
```

```
                  State          City  \
0      Alabama[edit]\n        Auburn
1      Alabama[edit]\n      Florence
2      Alabama[edit]\n  Jacksonville
3      Alabama[edit]\n    Livingston
4      Alabama[edit]\n    Montevallo
..                 ...           ...
512  Wisconsin[edit]\n         River
513  Wisconsin[edit]\n       Stevens
514  Wisconsin[edit]\n      Waukesha
515  Wisconsin[edit]\n    Whitewater
516    Wyoming[edit]\n       Laramie

                                  University
0                         [(Auburn University)]
1                [(University of North Alabama)]
2              [(Jacksonville State University)]
3                 [(University of West Alabama)]
4                   [(University of Montevallo)]
..                                           ...
512    [(University of Wisconsinâ€“River Falls)]
513  [(University of Wisconsinâ€“Stevens Point)]
```

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.

```
import re
uniplaces = []
with open('./uniplaces.txt') as file:
    for line in file:
        if '[edit]' in line:
            state = line
        else:
            #print(re.findall(r'\(.*?\)', line)[0])
            city=line.split(' ')[0]
            university=re.findall(r'\(.*?\)', line)
            uniplaces.append((state, city, university))
print(uniplaces)


uniDF = pd.DataFrame(uniplaces, columns=['State', 'City', 'University'])
print(uniDF)
```

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.

```
import re
def getCleanedData(item):
    #print(item)
    if ' (' in item:
        return item[:item.find(' (')]
    elif '[' in item:
        return item[:item.find('[')]
    else:
        return item
uniDF =  uniDF.applymap(getCleanedData)
print(uniDF)
```

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

In [103]:

```
import re
def getCleanedData(item):
    #print(item)
    if ' (' in item:
        return item[:item.find(' (')]
    elif '[' in item:
        return item[:item.find('[')]
    else:
        return item
uniDF =  uniDF.applymap(getCleanedData)
print(uniDF)
```

```
         State         City                                University
0      Alabama       Auburn                       [(Auburn University)]
1      Alabama     Florence             [(University of North Alabama)]
2      Alabama  Jacksonville           [(Jacksonville State University)]
3      Alabama    Livingston              [(University of West Alabama)]
4      Alabama    Montevallo                [(University of Montevallo)]
..         ...           ...                                         ...
512  Wisconsin         River    [(University of Wisconsinâ€“River Falls)]
513  Wisconsin       Stevens  [(University of Wisconsinâ€“Stevens Point)]
514  Wisconsin      Waukesha                      [(Carroll University)]
515  Wisconsin    Whitewater     [(University of Wisconsinâ€“Whitewater)]
516    Wyoming       Laramie                   [(University of Wyoming)]

[517 rows x 3 columns]
```

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.

```
trimetDF = pd.read_csv('trimet.csv')

print(trimetDF)

tempDF = []

for idx, row in trimetDF.iterrows():
    num = row['OCCURRENCES']
    for i in range(num):
        tempDF.append(row)

trimetDF = trimetDF.append(tempDF)
print(trimetDF);
```

In [87]:
```
trimetDF = pd.read_csv('trimet.csv')

print(trimetDF)
```

```
      VEHICLE_BREADCRUMB_ID  VEHICLE_NUMBER EQUIPMENT_CLASS  \
0                4313659804            3411               B
1                4313659805            3411               B
2                4313659806            3411               B
3                4313659807            3411               B
4                4313659808            3411               B
...                     ...             ...             ...
1236             4313659800            3411               B
1237             4313659801            3411               B
1238             4313659802            3411               B
1239             4313659803            3411               B
1240             4311203411            2934               B

         ARRIVE_DATETIME        SERVICE_DATE  ARRIVE_TIME   LONGITUDE  \
0     29OCT2021:07:16:40  29OCT2021:00:00:00      26200.0 -122.799683
1     29OCT2021:07:16:45  29OCT2021:00:00:00      26205.0 -122.799692
2     29OCT2021:07:16:50  29OCT2021:00:00:00      26210.0 -122.799703
3     29OCT2021:07:16:55  29OCT2021:00:00:00      26215.0 -122.799717
4     29OCT2021:07:17:00  29OCT2021:00:00:00      26220.0 -122.799732
...                  ...                 ...         ...         ...
```

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?

In [84]:
```
tempDF = []
for idx, row in trimetDF.iterrows():
    num = row['OCCURRENCES']
    for i in range(num):
        tempDF.append(row)

trimetDF = trimetDF.append(tempDF)
print(trimetDF);
```

```
     VEHICLE_BREADCRUMB_ID VEHICLE_NUMBER EQUIPMENT_CLASS     ARRIVE_DATETIME  \
0              4313659804           3411               B  29OCT2021:07:16:40
1              4313659805           3411               B  29OCT2021:07:16:45
2              4313659806           3411               B  29OCT2021:07:16:50
3              4313659807           3411               B  29OCT2021:07:16:55
4              4313659808           3411               B  29OCT2021:07:17:00
...                   ...            ...             ...                 ...
1236           4313659800           3411               B  29OCT2021:07:16:20
1237           4313659801           3411               B  29OCT2021:07:16:25
1238           4313659802           3411               B  29OCT2021:07:16:30
1239           4313659803           3411               B  29OCT2021:07:16:35
1240           4311203411           2934               B  29OCT2021:15:08:33

           SERVICE_DATE ARRIVE_TIME   LONGITUDE  LATITUDE DWELL  DISTANCE  \
```

```
      VALID_FLAG LAST_USER      LAST_TIMESTAMP
0              Y     TRANS  31OCT2021:06:06:40
1              Y     TRANS  31OCT2021:06:06:40
2              Y     TRANS  31OCT2021:06:06:40
3              Y     TRANS  31OCT2021:06:06:40
4              Y     TRANS  31OCT2021:06:06:40
...          ...       ...                 ...
1236           Y     TRANS  31OCT2021:06:06:40
1237           Y     TRANS  31OCT2021:06:06:40
1238           Y     TRANS  31OCT2021:06:06:40
1239           Y     TRANS  31OCT2021:06:06:40
1240           Y     TRANS  30OCT2021:06:21:52

[3283 rows x 17 columns]
```

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.

```
trimetDF['VALID_FLAG'].isnull().values.any()
```

Out[91]:

True

In [92]:

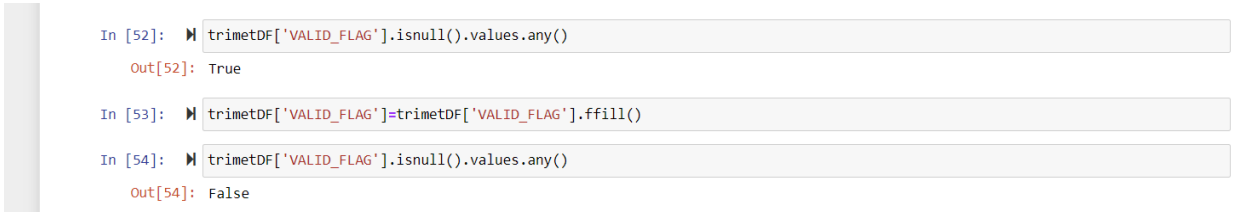
```
trimetDF['VALID_FLAG']=trimetDF['VALID_FLAG'].ffill()
```

In [93]:

```
trimetDF['VALID_FLAG'].isnull().values.any()
```

Out[93]:

False



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?

=> isNull() function checks bad data like NaN and duplicated() function checks for duplicates based on all columns.
We can find unique values in column using unique() function.
I think for VALID_FLAG expected value will be yes or no.

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.

```
trimetDF['ARRIVE_TIME'].isnull().values.any()
```

Out[94]:

True

In [97]:

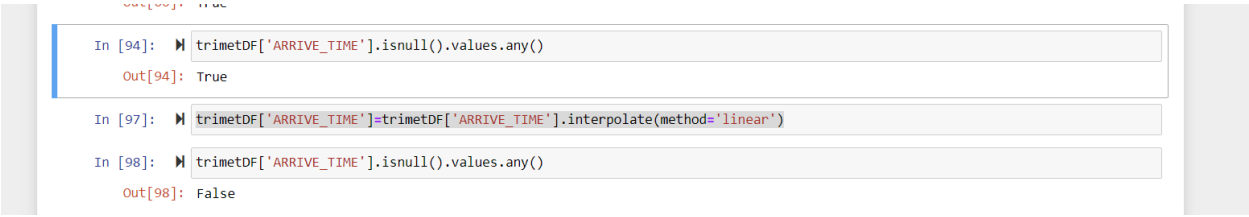
```
trimetDF['ARRIVE_TIME']=trimetDF['ARRIVE_TIME'].interpolate(method='linear')
```

In [98]:

```
trimetDF['ARRIVE_TIME'].isnull().values.any()
```

Out[98]:

False



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?
=> Interpolate treats the values as equally spaced so we can achieve that with an interpolate function.

Could you have used the **`interpolate()`** method for problem E above?
=> We can not use interpolate() method for above problem E, as dtype of “VALID_FLAG” columns is object and interpolation does not work on dtype object.

G. More Transformations

If you have finished all the previous transformations, try out those listed on this guide:
https://towardsdatascience.com/8-ways-to-transform-pandas-dataframes-b8c168ce878f

In [18]:
```
#Section G
df = pd.DataFrame({"names": ['Jane', 'John', 'Ashley', 'Max', 'Emily'], "A":[1,8,6,0,7], "B":[6,1,3,3,6], "C":[6,2,5,4,6], "D
                   "E":[1,8,7,8,6]})
df
```

Out[18]:

	names	A	B	C	D	E
0	Jane	1	6	6	9	1
1	John	8	1	2	8	8
2	Ashley	6	3	5	1	7
3	Max	0	3	4	0	8
4	Emily	7	6	6	0	6

In [19]:
```
#Add / drop columns
#The first and foremost way of transformation is adding or dropping columns. A new column can be added as follows:
df['new'] = np.random.random(5)
df
```

Out[19]:

	names	A	B	C	D	E	new
0	Jane	1	6	6	9	1	0.726613
1	John	8	1	2	8	8	0.660259
2	Ashley	6	3	5	1	7	0.799934
3	Max	0	3	4	0	8	0.236660
4	Emily	7	6	6	0	6	0.885093

In [20]:
```
df.drop('new', axis=1, inplace=True)
df
```

Out[20]:

	names	A	B	C	D	E
0	Jane	1	6	6	9	1
1	John	8	1	2	8	8
2	Ashley	6	3	5	1	7
3	Max	0	3	4	0	8
4	Emily	7	6	6	0	6

In [21]:
```
#Add / drop rows
#We can use the loc method to add a single row to a dataframe.
df.loc[5,:] = ['Jack', 3, 3, 4, 5, 1]
df
```

Out[21]:

	names	A	B	C	D	E
0	Jane	1.0	6.0	6.0	9.0	1.0
1	John	8.0	1.0	2.0	8.0	8.0
2	Ashley	6.0	3.0	5.0	1.0	7.0
3	Max	0.0	3.0	4.0	0.0	8.0
4	Emily	7.0	6.0	6.0	0.0	6.0
5	Jack	3.0	3.0	4.0	5.0	1.0

In [22]:
```
df.drop(5, axis=0, inplace=True)
df
```

Out[22]:

	names	A	B	C	D	E
0	Jane	1.0	6.0	6.0	9.0	1.0
1	John	8.0	1.0	2.0	8.0	8.0
2	Ashley	6.0	3.0	5.0	1.0	7.0
3	Max	0.0	3.0	4.0	0.0	8.0
4	Emily	7.0	6.0	6.0	0.0	6.0

In [23]:
```
#insert
df.insert(0, 'new', np.random.random(5))
df
```

Out[23]:

	new	names	A	B	C	D	E
0	0.675213	Jane	1.0	6.0	6.0	9.0	1.0
1	0.574648	John	8.0	1.0	2.0	8.0	8.0
2	0.003795	Ashley	6.0	3.0	5.0	1.0	7.0
3	0.859392	Max	0.0	3.0	4.0	0.0	8.0
4	0.245121	Emily	7.0	6.0	6.0	0.0	6.0

In [24]:
```
#melt
meltdf=pd.DataFrame({"names": ['Jane', 'John', 'Jack'], "day1":[1,5,4], "day2":[5,1,9], "day3":[8,3,8], "day4":[5,2,6],
                     "day5":[3,8,3]})
meltdf
```

Out[24]:

	names	day1	day2	day3	day4	day5
0	Jane	1	5	8	5	3
1	John	5	1	3	2	8
2	Jack	4	9	8	6	3

In [25]:
```
pd.melt(meltdf, id_vars='names').head()
```

Out[25]:

	names	variable	value
0	Jane	day1	1
1	John	day1	5
2	Jack	day1	4
3	Jane	day2	5
4	John	day2	1

In [27]:
```
#concat
df1=pd.DataFrame({"names": ['Jane', 'John', 'Jack'], "day1":[1,5,4], "day2":[5,1,9], "day3":[8,3,8], "day4":[5,2,6],
                  "day5":[3,8,3]})
df2=pd.DataFrame({"names": ['Max', 'Emily', 'Ashley'], "day1":[9,6,0], "day2":[0,8,2], "day3":[7,1,2], "day4":[4,9,4],
                  "day5":[5,0,0]})
print(df1)
print(df2)
```

```
  names  day1  day2  day3  day4  day5
0  Jane     1     5     8     5     3
1  John     5     1     3     2     8
2  Jack     4     9     8     6     3
    names  day1  day2  day3  day4  day5
0     Max     9     0     7     4     5
1   Emily     6     8     1     9     0
2  Ashley     0     2     2     4     0
```

In [29]:
```
pd.concat([df1, df2], axis=0, ignore_index=True)
```

Out[29]:

	names	day1	day2	day3	day4	day5
0	Jane	1	5	8	5	3
1	John	5	1	3	2	8
2	Jack	4	9	8	6	3
3	Max	9	0	7	4	5
4	Emily	6	8	1	9	0
5	Ashley	0	2	2	4	0

In [32]:
```
#Merge
customer=pd.DataFrame({"id":[1,2,3,4,5], "name": ['Jane', 'John', 'Jack', "Ashley", "Emily"], "ctg":['A', 'A', 'C', 'B', 'B']
order=pd.DataFrame({"id":[2,4,5,6], "amount": [24,32,25,44], "payment":["Credit card", "Credit card", "cash", "cash"]})
print(customer)
print(order)
```

```
   id    name ctg
0   1    Jane   A
1   2    John   A
2   3    Jack   C
3   4  Ashley   B
4   5   Emily   B
   id  amount      payment
0   2      24  Credit card
1   4      32  Credit card
2   5      25         cash
3   6      44         cash
```

In [33]:
```
customer.merge(order, on='id')
```

Out[33]:

	id	name	ctg	amount	payment
0	2	John	A	24	Credit card
1	4	Ashley	B	32	Credit card
2	5	Emily	B	25	cash

In [33]:
```
customer.merge(order, on='id')
```

Out[33]:

	id	name	ctg	amount	payment
0	2	John	A	24	Credit card
1	4	Ashley	B	32	Credit card
2	5	Emily	B	25	cash

In [35]:
```
#Get Dummies
dummydf=pd.DataFrame({"name": ['Jane', 'John', 'Jack', "Ashley", "Emily"], "ctg":['A', 'A', 'C', 'B', 'B'], "vals":[14.2, 21.
dummydf
```

Out[35]:

	name	ctg	vals
0	Jane	A	14.2
1	John	A	21.4
2	Jack	C	15.6
3	Ashley	B	12.1
4	Emily	B	17.7

In [37]:
```
pd.get_dummies(dummydf)
```

Out[37]:

	vals	name_Ashley	name_Emily	name_Jack	name_Jane	name_John	ctg_A	ctg_B	ctg_C
0	14.2	0	0	0	1	0	1	0	0
1	21.4	0	0	0	0	1	1	0	0

In [37]:
```
pd.get_dummies(dummydf)
```

Out[37]:

	vals	name_Ashley	name_Emily	name_Jack	name_Jane	name_John	ctg_A	ctg_B	ctg_C
0	14.2	0	0	0	1	0	1	0	0
1	21.4	0	0	0	0	1	1	0	0
2	15.6	0	0	1	0	0	0	0	1
3	12.1	1	0	0	0	0	0	1	0
4	17.7	0	1	0	0	0	0	1	0

In [38]:
```
pivotdf=pd.DataFrame({"name": ['Jane', 'John', 'John', 'John', 'Jane', 'Jane', 'Jane', 'John'], "ctg":['A', 'A', 'C', 'B', 'E
pivotdf
```

Out[38]:

	name	ctg	vals
0	Jane	A	14.2
1	John	A	21.4
2	John	C	15.6
3	John	B	12.1
4	Jane	B	17.7
5	Jane	C	12.5
6	Jane	C	8.6
7	John	B	19.1

In [39]:
```
pivotdf.pivot_table(index='name', columns='ctg', aggfunc='mean')
```

Out[39]:

	vals		
ctg	A	B	C
name			
Jane	14.2	17.7	10.55
John	21.4	15.6	15.60

For a longer guide on `melt` with a larger dataset:
https://towardsdatascience.com/transforming-data-in-python-with-pandas-melt-854221daf507

H. Transformation Visualizations

You can also visualize your data transformations with tools like: https://pandastutor.com/vis.html
Note however that you should provide a small sample of data like in the example they provide:

```
csv = '''
breed,type,longevity,size,weight
German Shepherd,herding,9.73,large,
Beagle,hound,12.3,small,
…
Maltese,toy,12.25,small,5.0
'''
```

You can then utilize this in your code as normal:

```
df = pd.read_csv(csv)
```