cw1-d-dataFrame-dataCleaning

March 15, 2022

[2]:
```
# np.nan or None can be used to represent missing values interpretable by pandas
# filtering out missing data
# dropna, fillna, isnull/notnull
import pandas as pd
import numpy as np
from numpy import nan as NA

ser = pd.Series([1,NA,3.5,None,7])
ser
```

[2]:
```
0    1.0
1    NaN
2    3.5
3    NaN
4    7.0
dtype: float64
```

[16]:
```
ser.isna().any()
```

[16]:
```
True
```

[2]:
```
ser.dropna()
```

[2]:
```
0    1.0
2    3.5
4    7.0
dtype: float64
```

[18]:
```
df = pd.DataFrame(np.random.randn(4,3))
df.loc[2] = None
df.iloc[0,2] = None
df
```

[18]:
```
          0         1         2
0 -0.203897 -0.179914       NaN
1  0.996920  2.196078  0.476274
2       NaN       NaN       NaN
3 -0.039568  0.061975  1.227356
```

[20]:
```
# check which columns have NaN
df.isna()
```

[20]:
```
       0      1      2
0  False  False   True
1  False  False  False
2   True   True   True
3  False  False  False
```

[22]:
```
# for large data
df.isna().any()
# returns names of columns
```

[22]:
```
0    True
1    True
2    True
dtype: bool
```

[27]:
```
# the same for rows
df.T.isna().any()
```

[27]:
```
0     True
1    False
2     True
3    False
dtype: bool
```

[28]:
```
# count rows with na
naRows = df.T.isna().any()
len(naRows[naRows == True])
```

[28]:
```
2
```

[29]:
```
# dropna by default removes rows, use axis= 'columns' or axis=1 to drop columns
df,df.dropna(), df.dropna(how='all'), df.dropna(thresh=1),df.dropna(axis =␣
↪1,thresh=2)# thresh doesn't work on cols?
```

[29]:
```
(          0         1         2
 0 -0.203897 -0.179914       NaN
 1  0.996920  2.196078  0.476274
 2       NaN       NaN       NaN
 3 -0.039568  0.061975  1.227356,
           0         1         2
 1  0.996920  2.196078  0.476274
 3 -0.039568  0.061975  1.227356,
           0         1         2
 0 -0.203897 -0.179914       NaN
```

```
1  0.996920  2.196078  0.476274
3 -0.039568  0.061975  1.227356,
          0         1         2
0 -0.203897 -0.179914       NaN
1  0.996920  2.196078  0.476274
3 -0.039568  0.061975  1.227356,
          0         1         2
0 -0.203897 -0.179914       NaN
1  0.996920  2.196078  0.476274
2       NaN       NaN       NaN
3 -0.039568  0.061975  1.227356)
```

[4]:
```
# filling in the missing values with fillna (fillna returns a new object, but␣
 ↪"inplace = True" may be used)
#df.fillna(0,inplace = True)
#df
df.fillna(0)
```

[4]:
```
          0         1         2
0 -0.683397  0.225693  0.000000
1 -0.294527 -1.000711  0.822506
2  0.000000  0.000000  0.000000
3  0.650876  0.645691 -1.581760
```

[5]:
```
# different replacement for each column with dictionary
df.fillna({0:1,2:3})
```

[5]:
```
          0         1         2
0 -0.683397  0.225693  3.000000
1 -0.294527 -1.000711  0.822506
2  1.000000       NaN  3.000000
3  0.650876  0.645691 -1.581760
```

[6]:
```
df.fillna(method='ffill'),df.fillna(method='ffill',axis = 1)
```

[6]:
```
(          0         1         2
 0 -0.683397  0.225693       NaN
 1 -0.294527 -1.000711  0.822506
 2 -0.294527 -1.000711  0.822506
 3  0.650876  0.645691 -1.581760,
          0         1         2
 0 -0.683397  0.225693  0.225693
 1 -0.294527 -1.000711  0.822506
 2       NaN       NaN       NaN
 3  0.650876  0.645691 -1.581760)
```

[7]:
```
# filling Series with mean
ser2 = pd.Series([None,2,None,1,5])
ser2, ser2.fillna(ser2.mean())
```

[7]:
```
(0    NaN
 1    2.0
 2    NaN
 3    1.0
 4    5.0
 dtype: float64,
 0    2.666667
 1    2.000000
 2    2.666667
 3    1.000000
 4    5.000000
 dtype: float64)
```

[9]:
```
# similar can be used in dataframes
df, df.fillna({0:df[0].mean()})
```

[9]:
```
(          0         1         2
 0 -0.683397  0.225693       NaN
 1 -0.294527 -1.000711  0.822506
 2       NaN       NaN       NaN
 3  0.650876  0.645691 -1.581760,
           0         1         2
 0 -0.683397  0.225693       NaN
 1 -0.294527 -1.000711  0.822506
 2 -0.109016       NaN       NaN
 3  0.650876  0.645691 -1.581760)
```

[ ]:
```
# arguments of fillna: value (or dict), method (ffill, bfill, etc.), axis,␣
 ↪inplace, limit
```

[16]:
```
# removing duplicate rows
df1 = pd.DataFrame({'v1':['a','b','c']*3, 'v2':[1,2]*4 + [1]})
df1, df1.duplicated()
```

[16]:
```
(  v1  v2
 0  a   1
 1  b   2
 2  c   1
 3  a   2
 4  b   1
 5  c   2
 6  a   1
 7  b   2
```

```
 8  c   1,
 0    False
 1    False
 2    False
 3    False
 4    False
 5    False
 6     True
 7     True
 8     True
 dtype: bool)
```

[19]:
```
df1.drop_duplicates()
```

[19]:
```
   v1  v2
0  a   1
1  b   2
2  c   1
3  a   2
4  b   1
5  c   2
```

[21]:
```
# restrict duplicate detection only to some columns
df1.drop_duplicates(['v1'])
#keeps the first occurence, keep = 'last' keeps last
```

[21]:
```
   v1  v2
0  a   1
1  b   2
2  c   1
```

[36]:
```
# transforming data with functions or mapping
df2 = pd.DataFrame({'city':['Warszawa','Lublin','Grodzisk Maz.
 ↪','Kraków','Poznań','Łódź','Pruszków','Gdańsk','Wrocław'],
                     'pop (thousand)':[1790,337,32,780,532,667,62,+470,642]})
df2
```

[36]:
```
             city  pop (thousand)
0       Warszawa            1790
1         Lublin             337
2  Grodzisk Maz.              32
3         Kraków             780
4         Poznań             532
5           Łódź             667
6       Pruszków              62
7         Gdańsk             470
8        Wrocław             642
```

[37]:
```
# generalisation by mapping
# example: mapping from city to voivodship
# with df.map(dictionary)
cityToVoivodship = {'Warszawa': 'Mazowieckie','Kraków':'Małopolskie','Poznań':
 ↪'Wielkoposkie','Lublin':'Lubelskie',
                    'Łódź':'Łódzkie','Gdańsk':'Pomorskie','Wrocław':
 ↪'Dolnośląskie','Grodzisk Maz.':'Mazowieckie','Pruszków':'Mazowieckie'}
df2['voivodship'] = df2['city'].map(cityToVoivodship)
df2
```

[37]:

	city	pop (thousand)	voivodship
0	Warszawa	1790	Mazowieckie
1	Lublin	337	Lubelskie
2	Grodzisk Maz.	32	Mazowieckie
3	Kraków	780	Małopolskie
4	Poznań	532	Wielkoposkie
5	Łódź	667	Łódzkie
6	Pruszków	62	Mazowieckie
7	Gdańsk	470	Pomorskie
8	Wrocław	642	Dolnośląskie

[45]:
```
# mapping with map(function)
df2['pop (million)'] = df2['pop (thousand)'].map(lambda x: x/1000)
df2
```

[45]:

	city	pop (thousand)	voivodship	pop (million)
0	Warszawa	1790	mazowieckie	1.790
1	Lublin	337	lubelskie	0.337
2	Grodzisk Maz.	32	mazowieckie	0.032
3	Kraków	780	małopolskie	0.780
4	Poznań	532	wielkoposkie	0.532
5	Łódź	667	łódzkie	0.667
6	Pruszków	62	mazowieckie	0.062
7	Gdańsk	470	pomorskie	0.470
8	Wrocław	642	dolnośląskie	0.642

[46]:
```
# string manipulation (e.g. lowercase)
df2['voivodship']=df2['voivodship'].str.lower()
df2
```

[46]:

	city	pop (thousand)	voivodship	pop (million)
0	Warszawa	1790	mazowieckie	1.790
1	Lublin	337	lubelskie	0.337
2	Grodzisk Maz.	32	mazowieckie	0.032
3	Kraków	780	małopolskie	0.780
4	Poznań	532	wielkoposkie	0.532
5	Łódź	667	łódzkie	0.667

```
6       Pruszków              62   mazowieckie          0.062
7         Gdańsk             470     pomorskie          0.470
8        Wrocław             642  dolnośląskie          0.642
```

[47]:
```
# value replacement with df.replace(dict) or df.replace(list1,list2)
df2.replace({'mazowieckie':'stołeczne','wielkoposkie':'wielkopolskie'})
df2
```

[47]:
```
            city  pop (thousand)    voivodship  pop (million)
0       Warszawa            1790   mazowieckie          1.790
1         Lublin             337     lubelskie          0.337
2  Grodzisk Maz.              32   mazowieckie          0.032
3         Kraków             780   małopolskie          0.780
4         Poznań             532  wielkoposkie          0.532
5           Łódź             667       łódzkie          0.667
6       Pruszków              62   mazowieckie          0.062
7         Gdańsk             470     pomorskie          0.470
8        Wrocław             642  dolnośląskie          0.642
```

[49]:
```
df2.replace(['mazowieckie','wielkoposkie'],['stołeczne','poznańskie'])
```

[49]:
```
            city  pop (thousand)    voivodship  pop (million)
0       Warszawa            1790     stołeczne          1.790
1         Lublin             337     lubelskie          0.337
2  Grodzisk Maz.              32     stołeczne          0.032
3         Kraków             780   małopolskie          0.780
4         Poznań             532    poznańskie          0.532
5           Łódź             667       łódzkie          0.667
6       Pruszków              62     stołeczne          0.062
7         Gdańsk             470     pomorskie          0.470
8        Wrocław             642  dolnośląskie          0.642
```

[52]:
```
# renaming index and columns with rename (index/columns = transforming function)
df2.rename(index = lambda x: x*10, columns = str.title)
```

[52]:
```
            City  Pop (Thousand)    Voivodship  Pop (Million)
0       Warszawa            1790   mazowieckie          1.790
10        Lublin             337     lubelskie          0.337
20 Grodzisk Maz.              32   mazowieckie          0.032
30        Kraków             780   małopolskie          0.780
40        Poznań             532  wielkoposkie          0.532
50          Łódź             667       łódzkie          0.667
60      Pruszków              62   mazowieckie          0.062
70        Gdańsk             470     pomorskie          0.470
80       Wrocław             642  dolnośląskie          0.642
```

[61]:
```
# discretization and binning
populationBins = [50,100,500,1000]
cats = pd.cut(list(df2['pop (thousand)']),populationBins)
cats.codes
# notice -1 for values outside the scope of bins
```

[61]:
```
array([-1,  1, -1,  2,  2,  2,  0,  1,  2], dtype=int8)
```

[80]:
```
correctedBins = [0,50,100,500,1000,float("inf")]
popCats = ['<50','<100','<500','<1000','over 1000']
cats = pd.cut(list(df2['pop (thousand)']),correctedBins)
cats
```

[80]:
```
[(1000.0, inf], (100.0, 500.0], (0.0, 50.0], (500.0, 1000.0], (500.0, 1000.0],
(500.0, 1000.0], (50.0, 100.0], (100.0, 500.0], (500.0, 1000.0]]
Categories (5, interval[float64, right]): [(0.0, 50.0] < (50.0, 100.0] < (100.0,
500.0] < (500.0, 1000.0] < (1000.0, inf]]
```

[77]:
```
cats.codes
```

[77]:
```
array([4, 2, 0, 3, 3, 3, 1, 2, 3], dtype=int8)
```

[81]:
```
df2.replace(df2['pop category'],pd.Series(cats.codes))
df2['pop category'] = df2['pop category'].replace(range(5),list(popCats))
df2
```

[81]:

	city	pop (thousand)	voivodship	pop (million)	pop category
0	Warszawa	1790	mazowieckie	1.790	over 1000
1	Lublin	337	lubelskie	0.337	<500
2	Grodzisk Maz.	32	mazowieckie	0.032	<50
3	Kraków	780	małopolskie	0.780	<1000
4	Poznań	532	wielkoposkie	0.532	<1000
5	Łódź	667	łódzkie	0.667	<1000
6	Pruszków	62	mazowieckie	0.062	<100
7	Gdańsk	470	pomorskie	0.470	<500
8	Wrocław	642	dolnośląskie	0.642	<1000

[75]:
```
pd.value_counts(cats)
```

[75]:
```
(500.0, 1000.0]    4
(100.0, 500.0]     2
(0.0, 50.0]        1
(50.0, 100.0]      1
(1000.0, inf]      1
dtype: int64
```