Data Cleaning in Python

For any data analysis tasks, data quality is crucial. "Garbage in, means garbage out!"

The second step, after exploring the data, is data cleaning.

Common data issues are:

- Missing values
- Outliers
- Invalid data
- Duplicated records
- Inconsistent values (or units)

How to deal with data issues:

- Drop

```
- pd.dropna()
- pd.dropna(axis=1)
- pd.dropna(how='any')
- pd.dropna(how='all')  drops rows where all their values are missing
```

- Replace

```
- fillna(np.mean())
- fillna(method='ffill')  *note first row will not be affected*
- fillna(method='bfill') *note last row will not be affected*
- fillna(mthod = 'ffill', axis = 1)
```

- Split strings

```
- .str.split('_', expand=True)
```

In this notebook, I will summarize various ways to inspect for data issues and resolve them.

In [7]:
```python
import numpy as np
import pandas as pd
```

Missing Values, NaN

To check for No value in numpy:

np.isnan(array_name)
np.isfinite checks

in pandas:

pd.isnull()
pd.notnull()

Note that in numpy nan is like a virus, any operations with nan yields nan. Here are some examples to illustrate:

In [2]:
```
3+np.nan
```
Out[2]:
```
nan
```

In [4]:
```
a= np.array([1,2,3, np.nan, np.nan, 4])
```

In [5]:
```
np.isnan(a)
```
Out[5]:
```
array([False, False, False,  True,  True, False])
```

In [6]:
```
a.sum()
```
Out[6]:
```
nan
```

In [13]:
```
np.isfinite(a)
```
Out[13]:
```
array([ True,  True,  True, False, False,  True])
```

In [16]:
```
pd.isnull(np.nan)
```
Out[16]:
```
True
```

In [17]:
```
pd.isnull(None)
```
Out[17]:
```
True
```

In [18]:
```
pd.isna(None)
```
Out[18]:
```
True
```

In [19]:
```
pd.notnull(None)
```
Out[19]:
```
False
```

In [21]:
```
pd.notna(None)
```
Out[21]:
```
False
```

Operations with Missing Values

In [8]:
```
a= np.array([1,2,3,np.nan, np.nan, 4])
```

In [12]:
```
pd.Series(a).count()
```

Out[12]:
```
4
```

In [14]:
```
pd.isnull(a)
```

Out[14]:
```
array([False, False, False,  True,  True, False])
```

In [15]:
```
a.sum()
```

Out[15]:
```
nan
```

In [16]:
```
pd.Series(a).sum()
```

Out[16]:
```
10.0
```

Note that numpy array needs to be turned into pandas series to get the value along with its index.

In [17]:
```
a[pd.notnull(a)]   # no index, just the values
```

Out[17]:
```
array([1., 2., 3., 4.])
```

In [18]:
```
s=pd.Series(a)
```

In [19]:
```
s
```

Out[19]:
```
0    1.0
1    2.0
2    3.0
3    NaN
4    NaN
5    4.0
dtype: float64
```

In [20]:
```
s[pd.notnull(s)]   # value plus index
```

Out[20]:
```
0    1.0
1    2.0
2    3.0
5    4.0
dtype: float64
```

In [22]:
```
s[pd.isnull(s)]
```
Out[22]:
```
3    NaN
4    NaN
dtype: float64
```

Droping Null values

pd.dropna()

In [23]:
```
s.dropna()
```
Out[23]:
```
0     1.0
1     2.0
2     3.0
5     4.0
dtype: float64
```

Droping null values on DataFrames

Dropping null values in pandas series is simple, but when it comes to DataFrames you need to consider what to drop? The entire columns or rows with missing values.

Defult dropna() will drop the entire row

dropna(axis=1) drops the columns with null values

In [26]:
```
df = pd.DataFrame({
    'Column A': [1, np.nan, 30, np.nan],
    'Column B': [2,8,31, np.nan],
    'Column C': [np.nan, 9, 32, 100],
    'Column D': [5, 8, 34, 110],
})
```

In [27]:
```
df
```
Out[27]:

	Column A	Column B	Column C	Column D
0	1.0	2.0	NaN	5
1	NaN	8.0	9.0	8
2	30.0	31.0	32.0	34
3	NaN	NaN	100.0	110

In [28]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
Column A    2 non-null float64
Column B    3 non-null float64
Column C    3 non-null float64
Column D    4 non-null int64
dtypes: float64(3), int64(1)
memory usage: 256.0 bytes
```

In [29]:
```
df.isnull()
```

Out[29]:

	Column A	Column B	Column C	Column D
0	False	False	True	False
1	True	False	False	False
2	False	False	False	False
3	True	True	False	False

In [30]:
```
df.isnull().sum()
```

Out[30]:
```
Column A    2
Column B    1
Column C    1
Column D    0
dtype: int64
```

In [31]:
```
df.dropna()
```

Out[31]:

	Column A	Column B	Column C	Column D
2	30.0	31.0	32.0	34

In [32]:
```
df.dropna(axis=1)
```

Out[32]:

	Column D
0	5
1	8
2	34
3	110

In [34]:
```
df.dropna(how='all') # drops rows where all values are missing, not just a few like in dropna()
```

Out[34]:

	Column A	Column B	Column C	Column D
0	1.0	2.0	NaN	5
1	NaN	8.0	9.0	8
2	30.0	31.0	32.0	34
3	NaN	NaN	100.0	110

Replacing Null Values

Instead of dropping null values, we might need to replace them with some other value. It can be replaced with a zero, the mean, or the closest value. What to do depends on the context.

pd.fillna(value)

In [35]:
```
s
```

Out[35]:
```
0    1.0
1    2.0
2    3.0
3    NaN
4    NaN
5    4.0
dtype: float64
```

In [36]:
```
s.fillna(0)
```

Out[36]:
```
0    1.0
1    2.0
2    3.0
3    0.0
4    0.0
5    4.0
dtype: float64
```

In [37]:
```
s.fillna(s.mean())
```

Out[37]:
```
0    1.0
1    2.0
2    3.0
3    2.5
4    2.5
5    4.0
dtype: float64
```

In [38]:
```
s
```

Out[38]:
```
0    1.0
1    2.0
2    3.0
3    NaN
4    NaN
5    4.0
dtype: float64
```

In [40]:
```
# fillna(method='ffill') fills NaN with the value before it
s.fillna(method='ffill')
```

Out[40]:
```
0    1.0
1    2.0
2    3.0
3    3.0
4    3.0
5    4.0
dtype: float64
```

In [41]:
```
# fillna(method='bfill') will fill with the value that comes after NaN
s.fillna(method='bfill')
```

Out[41]:
```
0    1.0
1    2.0
2    3.0
3    4.0
4    4.0
5    4.0
dtype: float64
```

In [42]:
```
df
```

Out[42]:

	Column A	Column B	Column C	Column D
0	1.0	2.0	NaN	5
1	NaN	8.0	9.0	8
2	30.0	31.0	32.0	34
3	NaN	NaN	100.0	110

In [43]:
```
df.fillna(method='ffill', axis=0)
```

Out[43]:

	Column A	Column B	Column C	Column D
0	1.0	2.0	NaN	5
1	1.0	8.0	9.0	8
2	30.0	31.0	32.0	34
3	30.0	31.0	100.0	110

In [44]:
```
df.fillna(method='ffill', axis=1)
```

Out[44]:

	Column A	Column B	Column C	Column D
0	1.0	2.0	2.0	5.0
1	NaN	8.0	9.0	8.0
2	30.0	31.0	32.0	34.0
3	NaN	NaN	100.0	110.0

In [45]:
```
#note that NaN values in the first column/row and last colunm/row were not aff
ected
```

In [46]:
```
s.dropna().count()
```

Out[46]: 4

Invalid Values

An example of invalide value is D for sex, or 200 for age.

To find these invalide values, we can use:
value_counts() method or
unique()

and once we find invalid values, we can replace them with **.replace()**

In [47]:
```
gender = pd.DataFrame({
    'Sex': ['M', 'F', 'F', 'D', '?'],
    'Age': [29, 30, 24, 290, 25],
})
```

In [48]:
```
gender
```

Out[48]:

	Sex	Age
0	M	29
1	F	30
2	F	24
3	D	290
4	?	25

In [49]:
```
gender['Sex'].unique()
```

Out[49]:
```
array(['M', 'F', 'D', '?'], dtype=object)
```

In [50]:
```
gender['Sex'].value_counts()
```

Out[50]:
```
F    2
D    1
?    1
M    1
Name: Sex, dtype: int64
```

In [51]:
```
gender['Sex'].replace('D', 'F')
```

Out[51]:
```
0    M
1    F
2    F
3    F
4    ?
Name: Sex, dtype: object
```

In [53]:
```
gender.replace({
    'Sex': {
        'D': 'F',
        'N': 'M'
    },
    'Age': {
        290: 29
    }
})
```

Out[53]:

	Sex	Age
0	M	29
1	F	30
2	F	24
3	F	29
4	?	25

In [54]:
```
gender[gender['Age'] > 100]
```

Out[54]:

	Sex	Age
3	D	290

In [55]:
```
gender.loc[gender['Age']>100,'Age'] = gender.loc[gender['Age']>100, 'Age']/10
```

In [56]:
```
gender
```

Out[56]:

	Sex	Age
0	M	29.0
1	F	30.0
2	F	24.0
3	D	29.0
4	?	25.0

Duplicate Values

simple use duplicated() and drop_duplicates()

In [57]:
```
players = pd.DataFrame({
    'Name': [
        'Kobe Bryant',
        'LeBron James',
        'Kobe Bryant',
        'Carmelo Anthny',
        'Kobe Bryant',
    ],
    'Position': [
        'SG',
        'SF',
        'SG',
        'SF',
        'SF'
    ]
})
```

In [58]:
```
players
```

Out[58]:

	Name	Position
0	Kobe Bryant	SG
1	LeBron James	SF
2	Kobe Bryant	SG
3	Carmelo Anthny	SF
4	Kobe Bryant	SF

In [59]:
```python
# In players we can see that Kobe is duplicated but with different positions
# what will duplicated() say
players.duplicated()
```

Out[59]:
```
0    False
1    False
2     True
3    False
4    False
dtype: bool
```

In [60]:
```python
# it noted that when column values change, it's a different recrod
```

In [63]:
```python
players.duplicated(subset=['Name'])
```

Out[63]:
```
0    False
1    False
2     True
3    False
4     True
dtype: bool
```

In [65]:
```python
players.duplicated(subset=['Name'], keep='last')
```

Out[65]:
```
0     True
1    False
2     True
3    False
4    False
dtype: bool
```

In [66]:
```python
players.drop_duplicates()
```

Out[66]:

	Name	Position
0	Kobe Bryant	SG
1	LeBron James	SF
3	Carmelo Anthny	SF
4	Kobe Bryant	SF

In [67]:
```python
players.drop_duplicates(subset=['Name'], keep='last')
```

Out[67]:

	Name	Position
1	LeBron James	SF
3	Carmelo Anthny	SF
4	Kobe Bryant	SF

In [68]:
```
datesplit = pd.DataFrame({
    'Date': [
        '1987_M_US _1',
        '1990?_M_UK_1',
        '1992_F_US_2',
        '1970?_M_  IT_1',
        '1985_F_I  T_2'
    ]
})
```

In [69]:
```
datesplit
```

Out[69]:

	Date
0	1987_M_US _1
1	1990?_M_UK_1
2	1992_F_US_2
3	1970?_M_ IT_1
4	1985_F_I T_2

In [72]:
```
datesplit['Date'].str.split('_')
```

Out[72]:
```
0       [1987, M, US , 1]
1       [1990?, M, UK, 1]
2        [1992, F, US, 2]
3     [1970?, M,   IT, 1]
4      [1985, F, I  T, 2]
Name: Date, dtype: object
```

In [77]:
```
datesplit=datesplit['Date'].str.split('_', expand=True)
```

In [78]:
```
datesplit.shape
```

Out[78]:
```
(5, 4)
```

In [79]:
```
datesplit.columns = ['Year', 'Sex', 'Country', 'Num of Children']
```

In [80]:
```
datesplit
```

Out[80]:

	Year	Sex	Country	Num of Children
0	1987	M	US	1
1	1990?	M	UK	1
2	1992	F	US	2
3	1970?	M	IT	1
4	1985	F	I T	2

In [81]:
```
datesplit['Year'].str.contains('\?')
```

Out[81]:
```
0    False
1     True
2    False
3     True
4    False
Name: Year, dtype: bool
```

In [82]:
```
datesplit['Country'].str.replace(' ', '')
```

Out[82]:
```
0    US
1    UK
2    US
3    IT
4    IT
Name: Country, dtype: object
```

In [ ]:
```

```