Reading: Reference guide: Data cleaning in Python

This reference guide contains common functions and methods that data professionals use to clean data. The reference guide contains three different tables of useful tools, each grouped by cleaning category: missing data, outliers, and label encoding.

Save this course item

You may want to save a copy of this guide for future reference. You can use it as a resource for additional practice or in your future professional projects. To access a downloadable version of this course item, click the link below and select "Use Template."

Reference guide: Data cleaning in Python

OR

If you don't have a Google account, you can download the item directly from the attachment below.

Reference guide: Data cleaning in Python

Missing data

The following pandas functions and methods are helpful when dealing with missing data.

df.info()

- **Description:** A **DataFrame** method that returns a concise summary of the dataframe, including a 'non-null count,' which helps you know the number of missing values

Example:

```
print(df)
print()
df.info()
```

Output:

```
   planet   radius_km  moons
0  Mercury  2440       0
1  Venus    6052       0
```

```
2       Earth         6371           1
3       Mars          3390           2
4      Jupiter        69911          80
5      Saturn         58232          83
6      Uranus          25362          27
7       Neptune       24622          14

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 3 columns):
planet      8 non-null object
radius_km    8 non-null int64
moons       8 non-null int64
dtypes: int64(2), object(1)
memory usage: 272.0+ bytes
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 3 columns):
planet      8 non-null object
radius_km    8 non-null int64
moons       8 non-null int64
dtypes: int64(2), object(1)
memory usage: 272.0+ bytes
None
```

[pd.isna() / pd.isnull()]

- **Description:** **pd.isna()** is a pandas function that returns a same-sized Boolean array indicating whether each value is null (you can also use **pd.isnull()** as an alias). Note that this function also exists as a **DataFrame** method.

  Example:

```
print(df)
print('\n After pd.isnull(): \n')

pd.isnull(df)
```

  Output:

```
      Planet        radius_km   moons
0     Mercury      2440          NaN
1     Venus        6052           NaN
2     Earth       6371           1.0
```

```
3     Mars       3390          NaN
4     Jupiter    69911         80.0
5     Saturn     58232         83.0
6     Uranus     25362         27.0
7     Neptune    24622         14.0

 After pd.isnull():

      Planet    radius_km    moons
0     False     False        True
1     False     False        True
2     False     False        False
3     False     False        True
4     False     False        False
5     False     False        False
6     False     False        False
7     False     False        False
```

[pd.notna() / pd.notnull()]

- **Description:** A pandas function that returns a same-sized Boolean array indicating whether each value is NOT null (you can also use **pd.notnull()** as an alias). Note that this function also exists as a **DataFrame** method.

Example:

```
print(df)
print('\n After notnull(): \n')
pd.notnull(df)
```

Output:

```
      Planet    radius_km    moons
0     Mercury   2440         NaN
1     Venus     6052         NaN
2     Earth     6371         1.0
3     Mars      3390         NaN
4     Jupiter   69911        80.0
5     Saturn    58232        83.0
6     Uranus    25362        27.0
7     Neptune   24622        14.0

 After notnull():
```

```
      Planet  radius_km  moons
0     True    True       False
1     True    True       False
2     True    True       True
3     True    True       False
4     True    True       True
5     True    True       True
6     True    True       True
7     True    True       True
```

[df.fillna()]

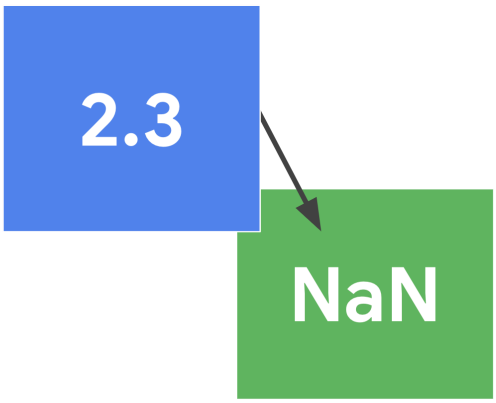
- **Description:** A **DataFrame** method that fills in missing values using specified method

Example:

```
print(df)
print('\n After fillna(): \n')

df.fillna(2)
```

Output:

```
      animal    class    color   legs
0     cardinal  Aves     red     NaN
1     gecko     Reptilia green   4.0
2     raven     Aves     black   NaN

 After fillna():

      animal    class    color   legs
0     cardinal  Aves     red     2.0
1     gecko     Reptilia green   4.0
2     raven     Aves     black   2.0
```



[df.replace()]

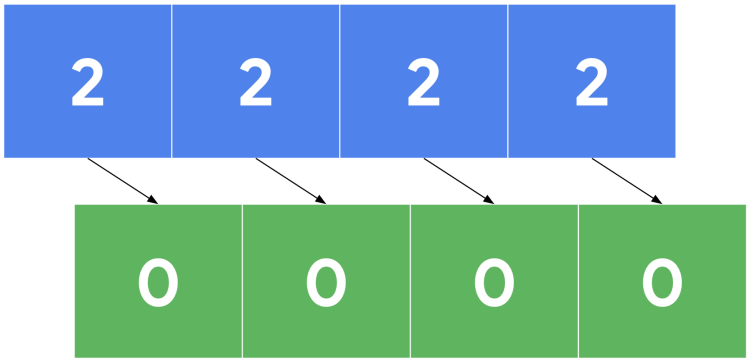
- **Description:** A **DataFrame** method that replaces specified values with other specified values. Can also be applied to pandas **Series**.

  Example:

```
print(df)
print('\n After replace(): \n')

df.replace('Aves', 'bird')
```

  Output:

```
        animal      class      color     legs
0       cardinal    Aves       red       2
1       gecko       Reptilia   green     4
2       raven       Aves       black     2

 After replace():

        animal     class      color     legs
0       cardinal   bird       red       2
1       gecko      Reptilia   green     4
2       raven      bird       black     2
```



[df.dropna()]

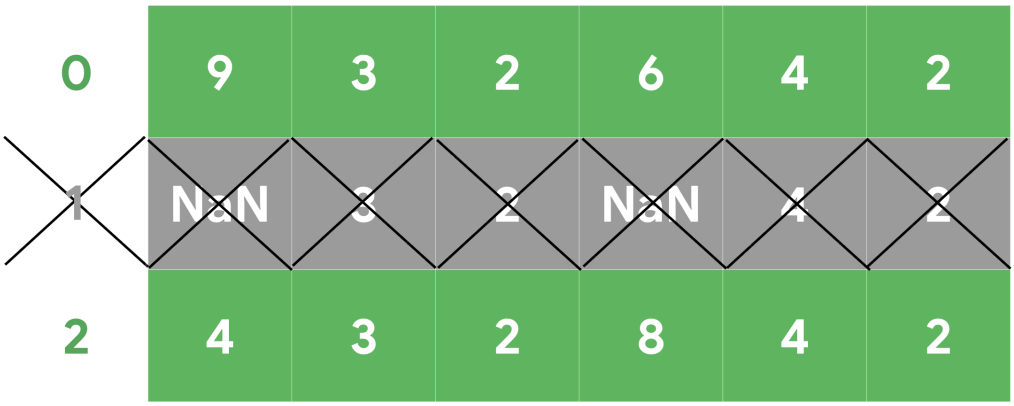
- **Description:** A **DataFrame** method that removes rows or columns that contain missing values, depending on the axis you specify.

Example:

```
print('Original df: \n \n', df)
print('\n After dropna(axis=0): \n')
print(df.dropna(axis=0))

print('\n After dropna(axis=1): \n')
print(df.dropna(axis=1))
```

Output:

```
Original df:

          animal      class      color    legs
0         NaN         Aves       red      2
1         gecko       Reptilia   green    4
2         raven       Aves       NaN      2

 After dropna(axis=0):

         animal     class          color      legs
1        gecko      Reptilia       green      4

 After dropna(axis=1):

       class        legs
0     Aves         2
1     Reptilia     4
2     Aves         2
```



Outliers

The following tools are helpful when dealing with outliers in a dataset.

[df.describe()]

- **Description:** A **DataFrame** method that returns general statistics about the dataframe which can help determine outliers

Example:

```
print(df)
print()
df.describe()
```

Output:

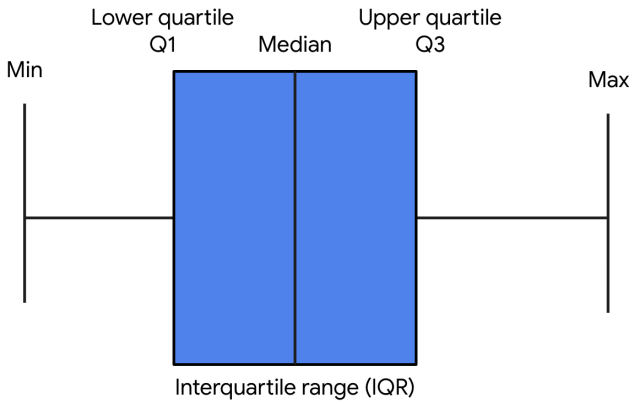
```
    planet      radius_km   moons
0   Mercury     2440        0
1   Venus       6052        0
2   Earth       6371        1
3   Mars        3390        2
4   Jupiter     69911       80
5   Saturn      58232       83
6   Uranus      25362       27
7   Neptune     24622       14

            radius_km           moons
count       8.000000            8.00000
mean        24547.500000        25.87500
std         26191.633528        35.58265
min         2440.000000         0.00000
25%         5386.500000         0.75000
```

```
50%     15496.500000     8.00000
75%     33579.500000     40.25000
max     69911.000000     83.00000
```

[sns.boxplot()]

- **Description:** A seaborn function that generates a box plot. Data points beyond 1.5x the interquartile range are considered outliers.

  Example:



Label encoding

The following tools are helpful when performing label encoding.

[df.astype()]

- **Description:** A **DataFrame** method that allows you to encode its data as a specified dtype. Note that this method can also be used on **Series** objects.

  Example:

```
print(df)
print('\n Original dtypes of df: \n')

print(df.dtypes)

print('\n dtypes after casting \'class\' column as categorical: \n')

df['class'] = df['class'].astype('category')

print(df.dtypes)
```

Output:

```
      animal       class      color     legs
0     cardinal     Aves       red       2
1     gecko        Reptilia   green     4
2     raven        Aves       black     2

 Original dtypes of df:

animal    object
class     object
color     object
legs      int64
dtype:    object

 dtypes after casting 'class' column as categorical:

animal     object
class      category
color      object
legs       int64
dtype:     object
```

[Series.cat.codes]

- **Description:** A **Series** attribute that returns the numeric category codes of the series.

Example:

```
# Cast 'class' column as categorical
df['class'] = df['class'].astype('category')

print('\n \'class\' column: \n')
print(df['class'])

print('\n Category codes of \'class\' column: \n')

df['class'].cat.codes
```

Output:

```
   'class' column:

0       Aves
1       Reptilia
```

```
2         Aves
Name: class, dtype: category
Categories (2, object): [Aves, Reptilia]

 Category codes of 'class' column:

0     0
1     1
2     0
dtype: int8
```

pd.get_dummies()

- **Description:** A function that converts categorical values into new binary columns—one for each different category

  Example:

index	rain
0	mild
1	mild
2	heavy
3	scattered
4	heavy
5	severe
6	severe
7	mild
8	heavy
9	scattered
10	scattered

index	rain_mild	rain_scattered	rain_heavy	rain_severe
0	1	0	0	0
1	1	0	0	0
2	0	0	1	0
3	0	1	0	0
4	0	0	1	1
5	0	0	0	1
6	0	0	0	1
7	1	0	0	0
8	0	0	1	0
9	0	1	0	0
10	0	1	0	0

LabelEncoder()

- **Description:** A transformer from **scikit-learn.preprocessing** that encodes specified categories or labels with numeric codes. Note that when building predictive models it should only be used on target variables (i.e., *y* data).

  Example:
  It can be used to normalize labels:

```
from sklearn.preprocessing import LabelEncoder

# Instantiate LabelEncoder()
encoder = LabelEncoder()

data = [1, 2, 2, 6]

# Fit to the data
```

```
encoder.fit(data)

# Transform the data
transformed = encoder.transform(data)

# Reverse the transformation
inverse = encoder.inverse_transform(transformed)

print('Data =', data)
print('\n Classes: \n', encoder.classes_)
print('\n Encoded (normalized) classes: \n', transformed)
print('\n Reverse from encoded classes to original: \n', inverse)
```

Output:

```
Data = [1, 2, 2, 6]

 Classes:
[1 2 6]

 Encoded (normalized) classes:
[0 1 1 2]

 Reverse from encoded classes to original:
[1 2 2 6]
```

It can be used to convert categorical labels into numeric:

```
from sklearn.preprocessing import LabelEncoder

# Instantiate LabelEncoder()
encoder = LabelEncoder()

data = ['paris', 'paris', 'tokyo', 'amsterdam']

# Fit to the data
encoder.fit(data)

# Transform the data
transformed = encoder.transform(data)

# New data
new_data = [0, 2, 1, 1, 2]
```

```
# Get classes of new data
inverse = encoder.inverse_transform(new_data)

print('Data =', data)
print('\n Classes: \n', list(encoder.classes_))
print('\n Encoded classes: \n', transformed)
print('\n New data =', new_data)
print('\n Convert new_data to original classes: \n', list(inverse))
```

Output:

```
Data = ['paris', 'paris', 'tokyo', 'amsterdam']

 Classes:
['amsterdam', 'paris', 'tokyo']

 Encoded classes:
[1 1 2 0]

 New data = [0, 2, 1, 1, 2]

 Convert new_data to original classes:
['amsterdam', 'tokyo', 'paris', 'paris', 'tokyo']
```

Key takeaways

There are many tools that data professionals can use to perform data cleaning on a wide range of data. The information you learn from missing data, outliers, and transforming categorical to numeric data will help you prepare datasets for further analysis throughout your career.
