DATA PREPARATION AND VISUALIZATION

Department of Mathematical Economics

National Economics University
https://www.neu.edu.vn/

Data Cleaning

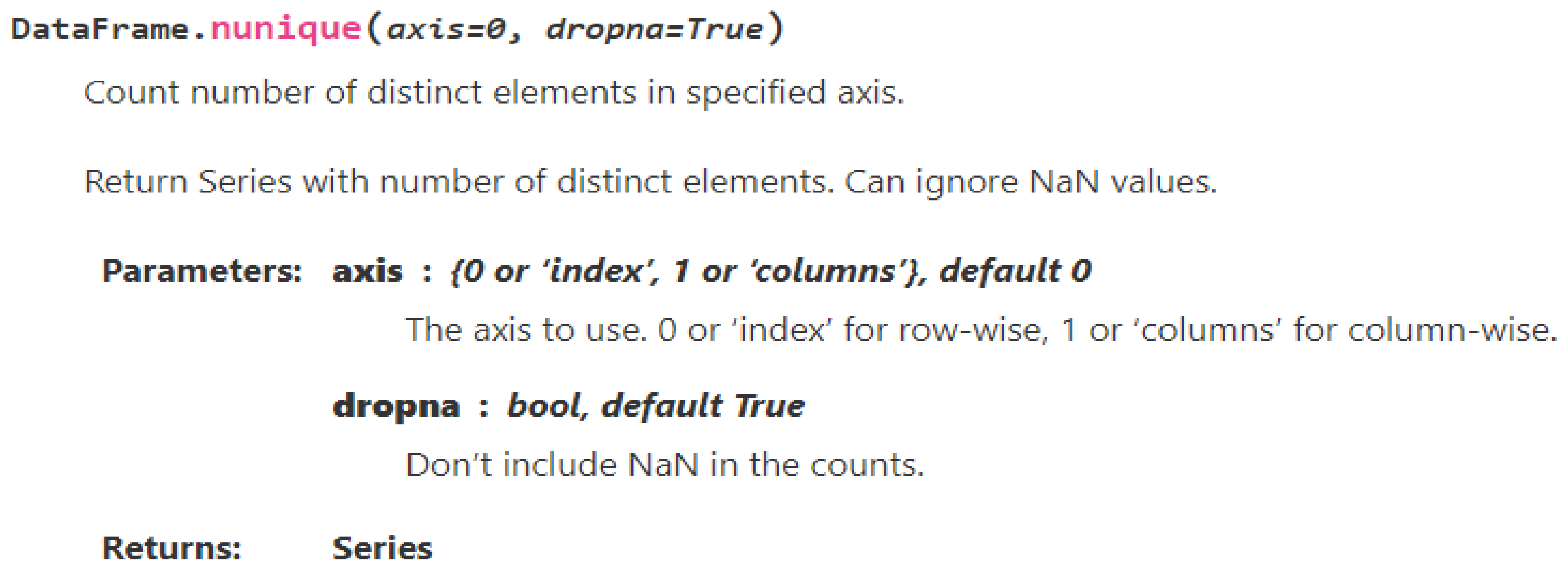
Outline

- Handling missing data
- Basic Data Cleaning
  - Identify Columns That Contain a Single Value
  - Delete Columns that Contain a Single Value
  - Remove Columns That Have A Low Variance
  - Identify Rows that Contain Duplicate Data
  - Delete Rows that Contain Duplicate Data
- Outlier Identification and Removal

Basic Data Cleaning

Essential Functions

- Columns that have a single value for all rows do not contain any information for modeling
- We can detect columns that have this property using the nunique() pandas function



```
DataFrame.nunique(axis=0, dropna=True)
```

Count number of distinct elements in specified axis.

Return Series with number of distinct elements. Can ignore NaN values.

Parameters: **axis** : ***{0 or 'index', 1 or 'columns'}, default 0***

The axis to use. 0 or 'index' for row-wise, 1 or 'columns' for column-wise.

dropna : ***bool, default True***

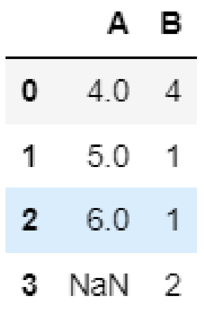
Don't include NaN in the counts.

Returns: **Series**

Basic Data Cleaning

Essential Functions

Df =



	A	B
0	4.0	4
1	5.0	1
2	6.0	1
3	NaN	2

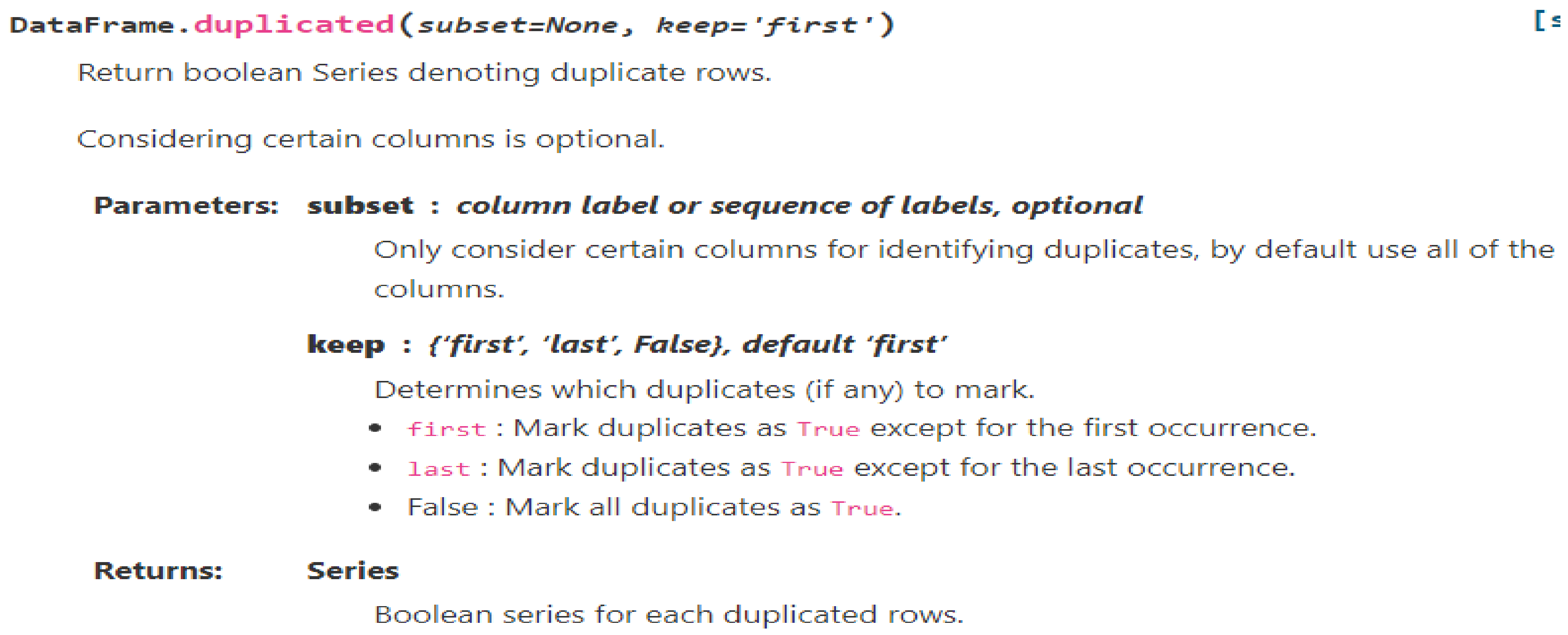
Df.nunique() = ???

Df.nunique(axis=1) = ???

Df.nunique(dropna = False)

Basic Data Cleaning

Essential Functions



```
DataFrame.duplicated(subset=None, keep='first')
```
[s

Return boolean Series denoting duplicate rows.

Considering certain columns is optional.

Parameters: **subset : *column label or sequence of labels, optional***

Only consider certain columns for identifying duplicates, by default use all of the columns.

keep : *{'first', 'last', False}, default 'first'*

Determines which duplicates (if any) to mark.

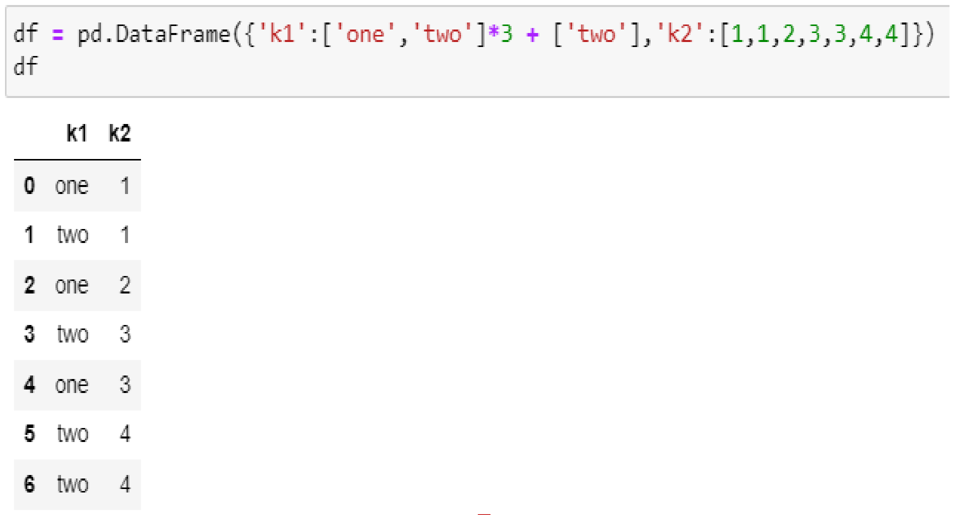
- `first` : Mark duplicates as `True` except for the first occurrence.
- `last` : Mark duplicates as `True` except for the last occurrence.
- False : Mark all duplicates as `True`.

Returns: **Series**

Boolean series for each duplicated rows.

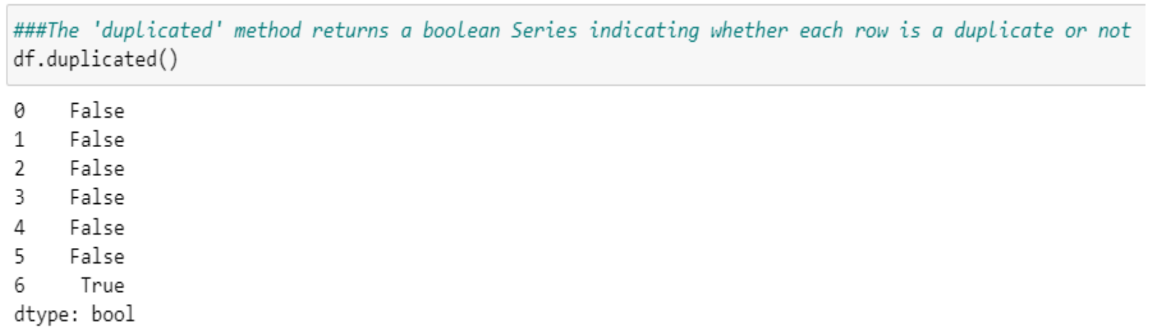
Basic Data Cleaning

Essential Functions



```
df = pd.DataFrame({'k1':['one','two']*3 + ['two'],'k2':[1,1,2,3,3,4,4]})
df
```

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4
6	two	4



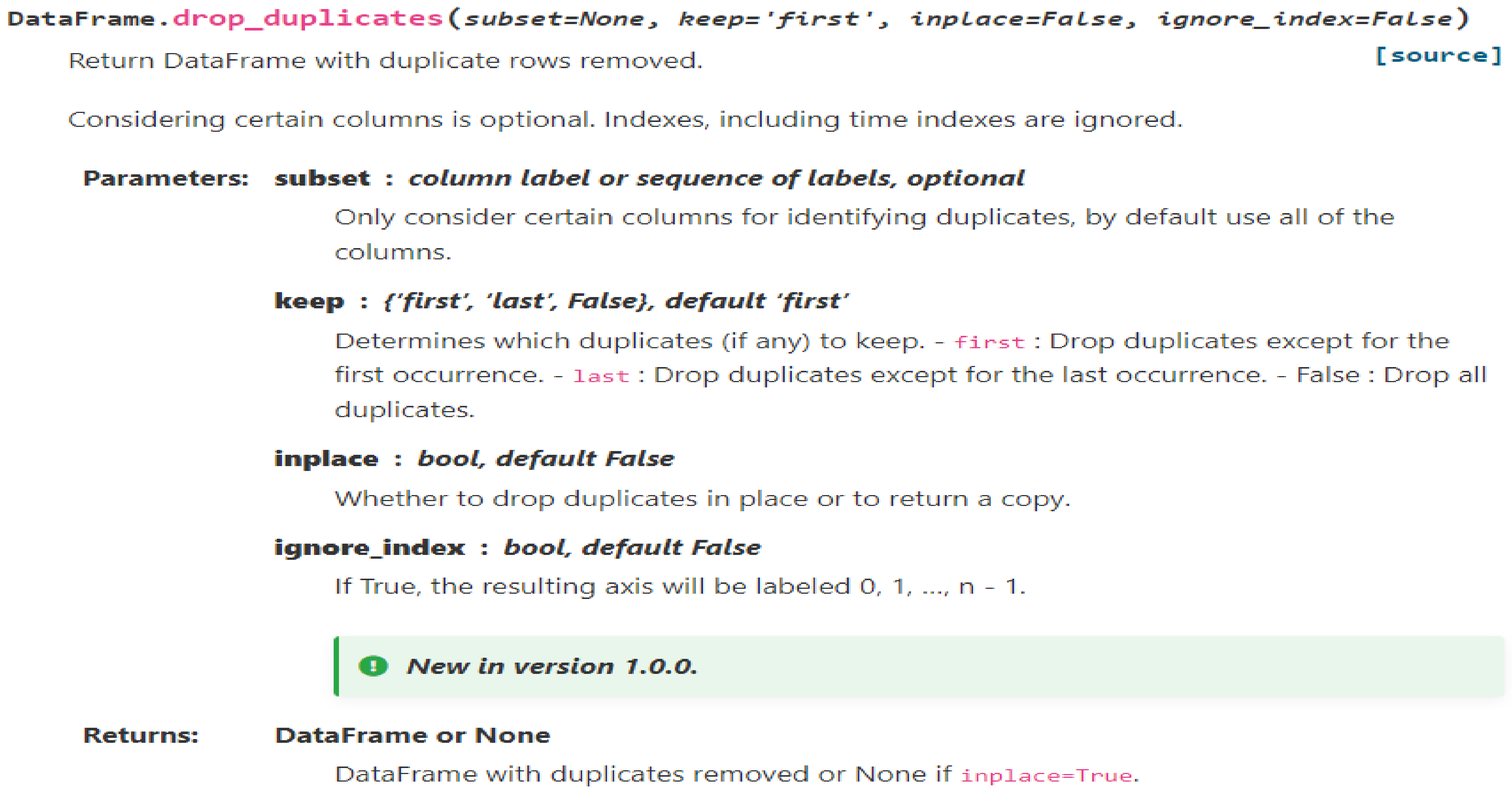
```
###The 'duplicated' method returns a boolean Series indicating whether each row is a duplicate or not
df.duplicated()
```

```
0    False
1    False
2    False
3    False
4    False
5    False
6     True
dtype: bool
```

- How to Delete Rows That Contain Duplicate Data?
- Pandas provides the drop_duplicates() function that achieves exactly this

Basic Data Cleaning

Essential Functions



DataFrame.drop_duplicates(*subset=None, keep='first', inplace=False, ignore_index=False*)

[source]

Return DataFrame with duplicate rows removed.

Considering certain columns is optional. Indexes, including time indexes are ignored.

Parameters: **subset** : ***column label or sequence of labels, optional***

Only consider certain columns for identifying duplicates, by default use all of the columns.

keep : ***{'first', 'last', False}, default 'first'***

Determines which duplicates (if any) to keep. - `first` : Drop duplicates except for the first occurrence. - `last` : Drop duplicates except for the last occurrence. - False : Drop all duplicates.

inplace : ***bool, default False***

Whether to drop duplicates in place or to return a copy.

ignore_index : ***bool, default False***

If True, the resulting axis will be labeled 0, 1, …, n - 1.

New in version 1.0.0.

Returns: **DataFrame or None**

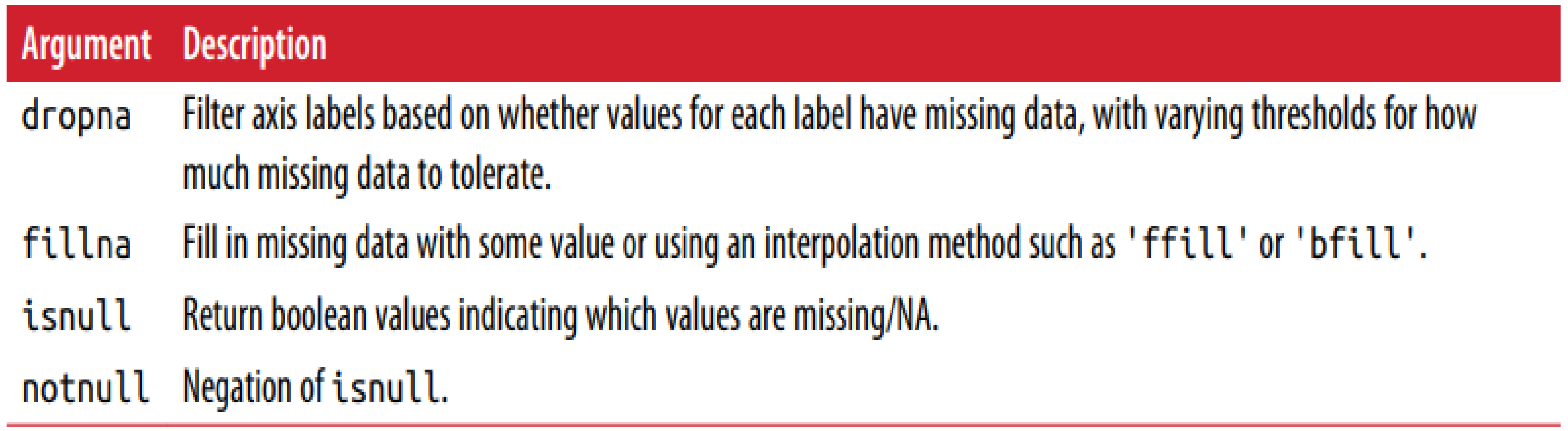
DataFrame with duplicates removed or None if `inplace=True`.

Handling Missing Data

Essential Functions

- For numeric data, pandas uses the floating-point value NaN(Not a Number) to present missing data
- All of the descriptive statistics on pandas objects exclude missing data by default
- The build-in Python None value is also treated as NA in object arrays

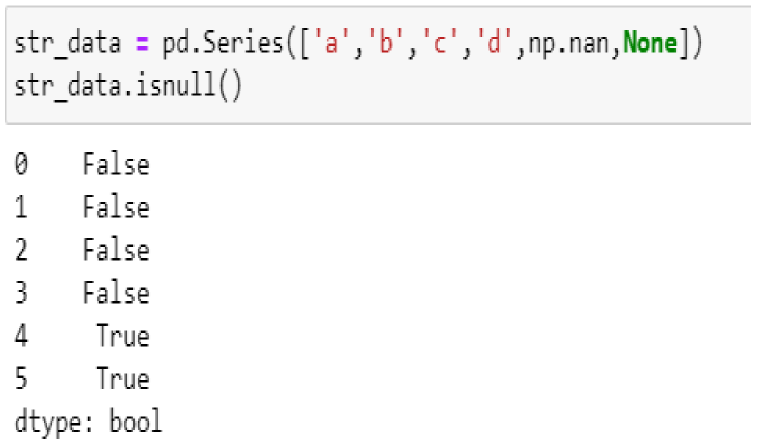
Table 7-1. NA handling methods



Argument	Description
`dropna`	Filter axis labels based on whether values for each label have missing data, with varying thresholds for how much missing data to tolerate.
`fillna`	Fill in missing data with some value or using an interpolation method such as `'ffill'` or `'bfill'`.
`isnull`	Return boolean values indicating which values are missing/NA.
`notnull`	Negation of `isnull`.

Handling Missing Data

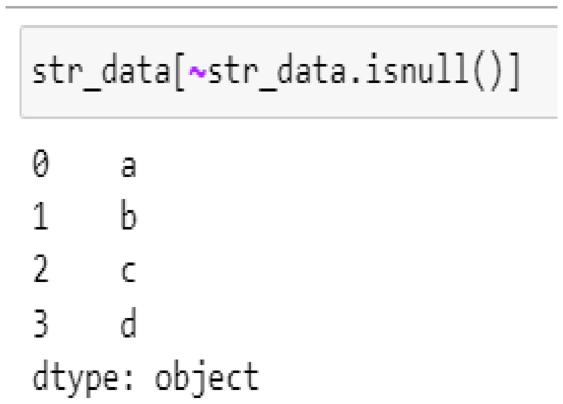
isnull function



```
str_data = pd.Series(['a','b','c','d',np.nan,None])
str_data.isnull()
```

```
0    False
1    False
2    False
3    False
4     True
5     True
dtype: bool
```

Filtering out missing values



```
str_data[~str_data.isnull()]
```

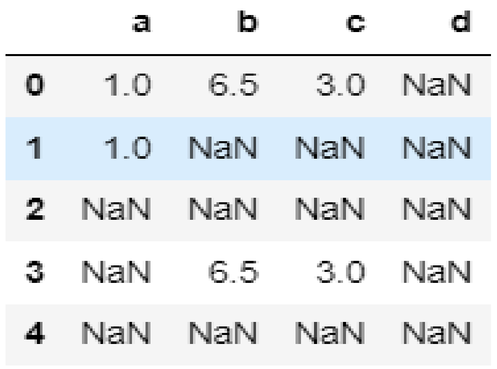
```
0    a
1    b
2    c
3    d
dtype: object
```

- We can use *str_data.isnull().sum()* to count the number of missing values in a series
- Question: How to calculate the number of missing values for each column or each row in a DataFrame?

Handling Missing Data

isnull function

- Question: How to calculate the number of missing values for each column or each row in a DataFrame?
  - Create the following DataFrame, named df



	a	b	c	d
0	1.0	6.5	3.0	NaN
1	1.0	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	6.5	3.0	NaN
4	NaN	NaN	NaN	NaN

- Try df.isnull().sum()
- Try df.isnull().sum(axis=1)

Handling Missing Data

isnull function

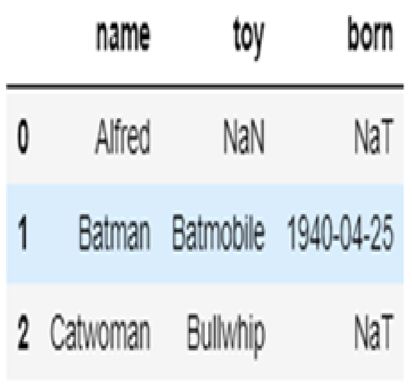
How to drop columns (or rows) with more than x% missing value

Handling Missing Data

Dropna Function

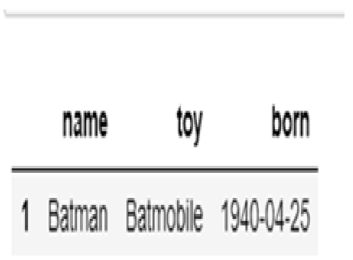
- Df.dropna() is used to remove columns or rows that contain missing values
- We can determine if rows or columns which contain missing values are removed using the axis parameters
- Ex:

df



	name	toy	born
0	Alfred	NaN	NaT
1	Batman	Batmobile	1940-04-25
2	Catwoman	Bullwhip	NaT

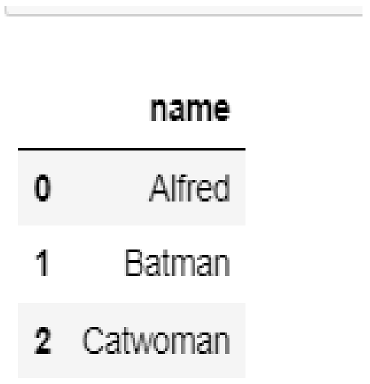
df.dropna()



	name	toy	born
1	Batman	Batmobile	1940-04-25

By default axis=0 or 'index'

df.dropna(axis=1)



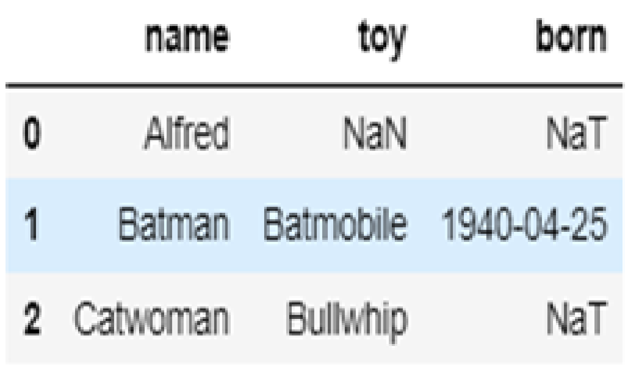
	name
0	Alfred
1	Batman
2	Catwoman

Handling Missing Data

Dropna Function

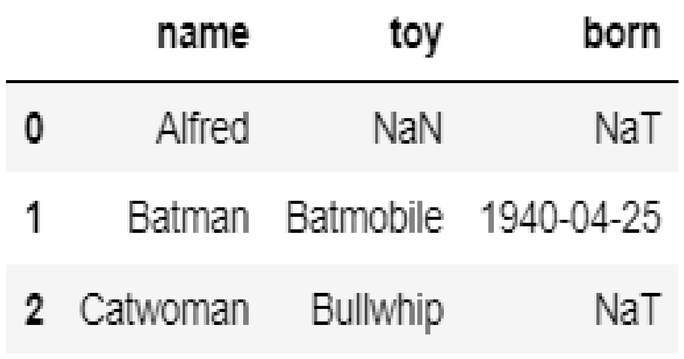
- We can also determine if rows or columns is removed from DataFrame, when we have at least one NA or all NA
  - ‘any’: if any NA values are present, drop that row or column
  - ‘all’: if all values are NA, drop that row or column
- Ex

df



	name	toy	born
0	Alfred	NaN	NaT
1	Batman	Batmobile	1940-04-25
2	Catwoman	Bullwhip	NaT

df.dropna(how = ‘all’)



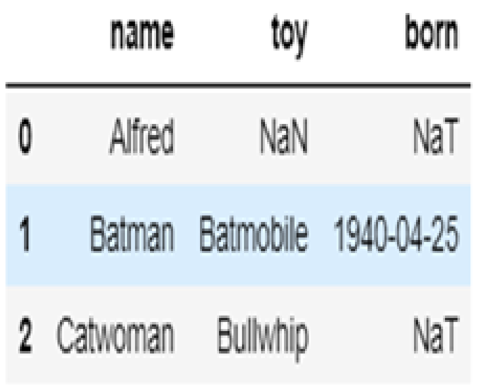
	name	toy	born
0	Alfred	NaN	NaT
1	Batman	Batmobile	1940-04-25
2	Catwoman	Bullwhip	NaT

Hanling Missing Data

Dropna Function

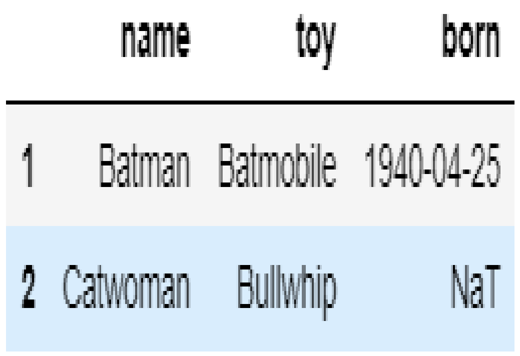
- The ‘thresh’ parameter tell us to keep only the rows(or columns) with at least *m* non-NA values
- We can define in which columns to look for missing values.
  - Ex: df.dropna(subset=[‘name’,’toy’])

df



	name	toy	born
0	Alfred	NaN	NaT
1	Batman	Batmobile	1940-04-25
2	Catwoman	Bullwhip	NaT

df.dropna(thresh=2)



	name	toy	born
1	Batman	Batmobile	1940-04-25
2	Catwoman	Bullwhip	NaT

Handling Missing Data

Dropna Function

Parameters	Description
Axis	Axis to drop, default axis=0
How	{'any','all'} default 'any', specify how the method will decide to drop a row(or column) from the DataFrame
Thresh	Require that many non-NA values
Subset	Labels along other axis to consider
Inplace	Bool, default False. Modify the calling object without producing a copy

Handling Missing Data

Dropna Function

Use the dropna method to drop columns that contain at least 90 percent null value

Handling Missing Data

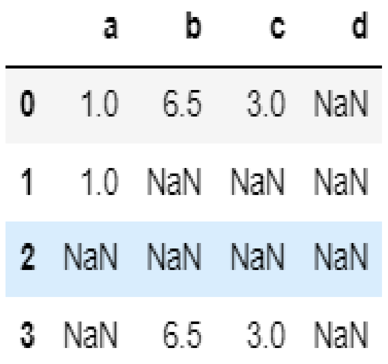
Fillna Function

- Rather than filtering out missing data (and potentially discarding other data along with it), you may want to fill in the "holes" in any number of ways
- For most purposes, the fillna method is the workhorse function to use
- Value to use to fill holes (e.g. 0), alternately a dict/Series/DataFrame of values specifying which value to use for each index (for a Series) or column (for a DataFrame).
    - Ex: age, number of transactions attributes
- Values not in the dict/Series/DataFrame will not be filled.
- This value cannot be a list.

Handling Missing Data

Fillna Function

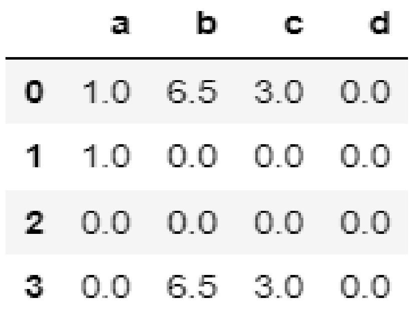
Df



	a	b	c	d
0	1.0	6.5	3.0	NaN
1	1.0	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	6.5	3.0	NaN

Scalar value

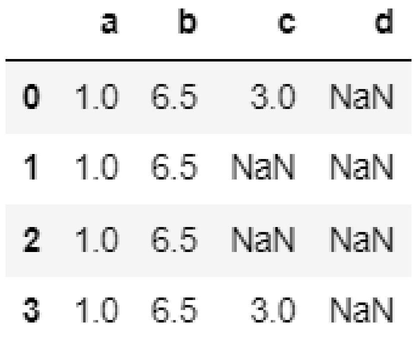
Df.fillna(0)



	a	b	c	d
0	1.0	6.5	3.0	0.0
1	1.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0
3	0.0	6.5	3.0	0.0

A dict
We can use a different fill value for each column

df.fillna({‘a’:1,’b’:6.5})

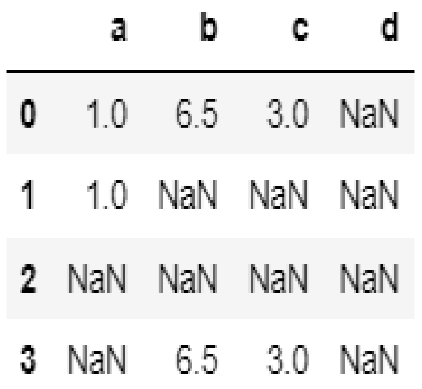


	a	b	c	d
0	1.0	6.5	3.0	NaN
1	1.0	6.5	NaN	NaN
2	1.0	6.5	NaN	NaN
3	1.0	6.5	3.0	NaN

Hanling Missing Data

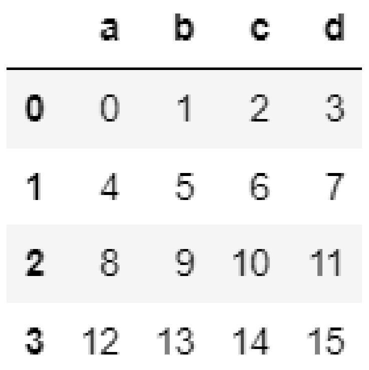
Fillna Function

df



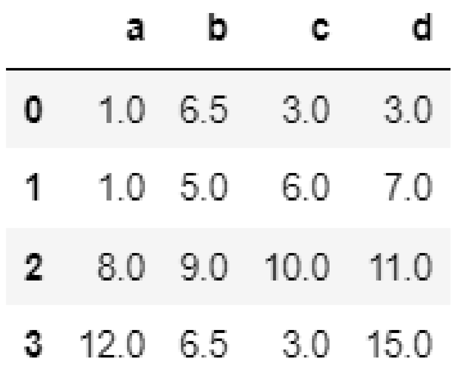
	a	b	c	d
0	1.0	6.5	3.0	NaN
1	1.0	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	6.5	3.0	NaN

df2



	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

df.fillna(df2)



	a	b	c	d
0	1.0	6.5	3.0	3.0
1	1.0	5.0	6.0	7.0
2	8.0	9.0	10.0	11.0
3	12.0	6.5	3.0	15.0

NOTE: When filling using a DataFrame, replacement happens along the same column names and same indices.

df.fillna(df2.drop(‘d’))???

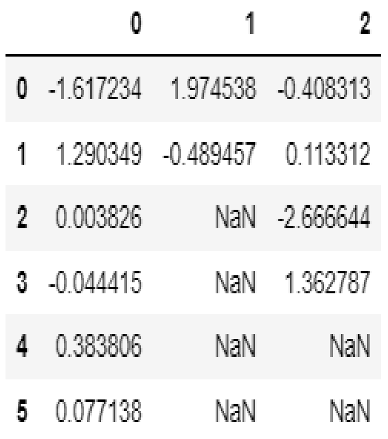
Handling Missing Data

Fillna Function

The interpolation methods can be used with fillna

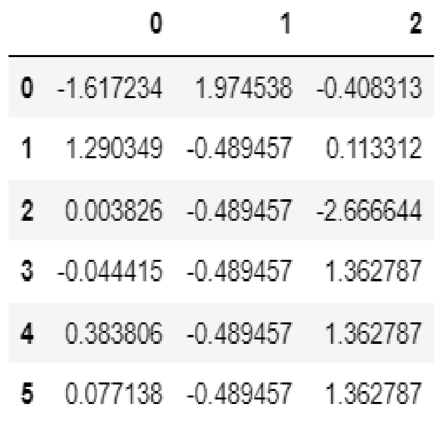
- If method = ‘ffill’: progagate last valid observation forward to next valid backfill
- If method = ‘bfill’: use next valid observation to fill gap
- We can use the limit parameter to determine the maximum number of consecutive NaN values to forward/backward fill

df



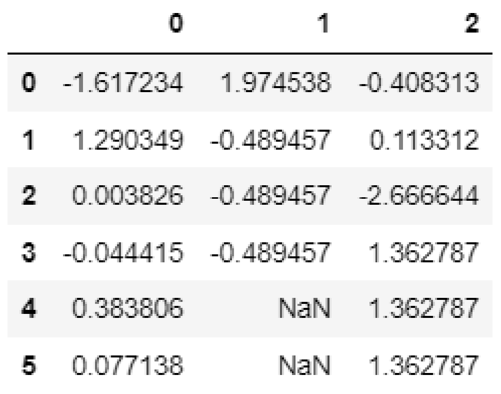
	0	1	2
0	-1.617234	1.974538	-0.408313
1	1.290349	-0.489457	0.113312
2	0.003826	NaN	-2.666644
3	-0.044415	NaN	1.362787
4	0.383806	NaN	NaN
5	0.077138	NaN	NaN

df.fillna(method=‘ffill’)



	0	1	2
0	-1.617234	1.974538	-0.408313
1	1.290349	-0.489457	0.113312
2	0.003826	-0.489457	-2.666644
3	-0.044415	-0.489457	1.362787
4	0.383806	-0.489457	1.362787
5	0.077138	-0.489457	1.362787

df.fillna(method=‘ffill’,limit = 2)



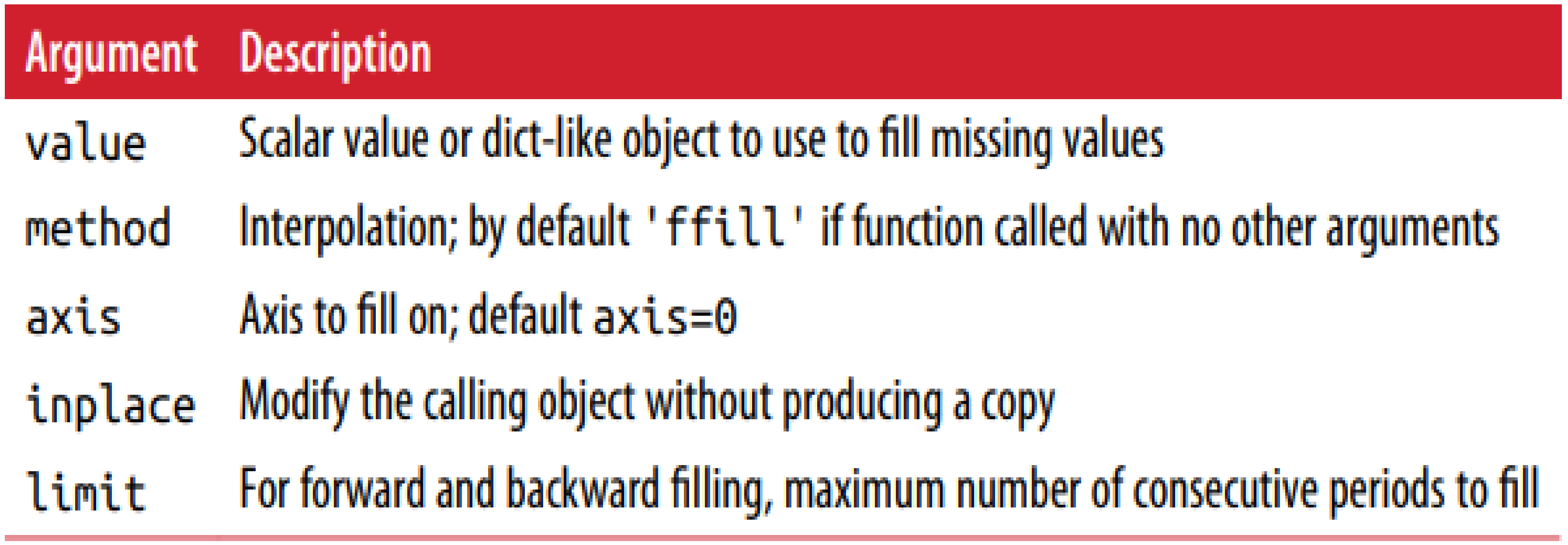
	0	1	2
0	-1.617234	1.974538	-0.408313
1	1.290349	-0.489457	0.113312
2	0.003826	-0.489457	-2.666644
3	-0.044415	-0.489457	1.362787
4	0.383806	NaN	1.362787
5	0.077138	NaN	1.362787

Question: df.fillna(method='bfill')???

Handling Missing Data

Fillna Function

Table 7-2. fillna function arguments



Argument	Description
`value`	Scalar value or dict-like object to use to fill missing values
`method`	Interpolation; by default `'ffill'` if function called with no other arguments
`axis`	Axis to fill on; default `axis=0`
`inplace`	Modify the calling object without producing a copy
`limit`	For forward and backward filling, maximum number of consecutive periods to fill

Statistical Imputation

- The missing data can be imputed by a suitable substitute and there are multiple strategies for it
  - Replace with mean
  - Replace with Median
  - Replace with Most Frequent Occuring
  - Replace with Constant
- Sklearn provides the SimpleImputer module that we can use to do this kind of task
- Install sklearn package in jupyter notebook: !pip install scikit-learn

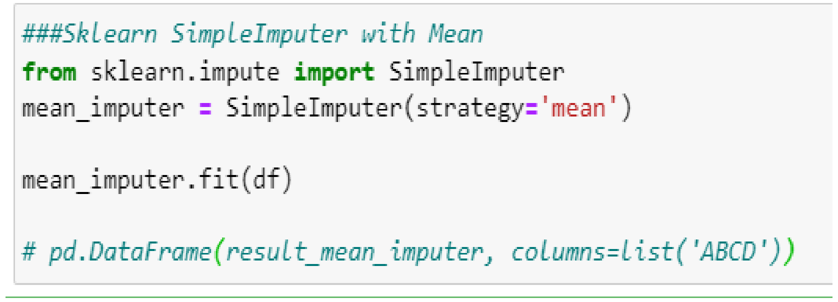
Statistical Imputation

SimpleImputer method

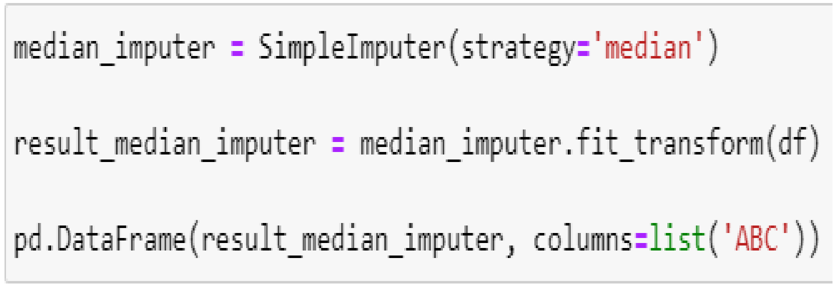
Parameters	Description
Missing_values	The placeholder for the missing values. It can be int, float, str, np.nan, None or pandas.NA, default = np.nan
strategy	Str, default = 'mean' - 'mean'(or median): replace missing values using the mean (median) along each column - 'most_frequent": replace missing value using the most frequent value along each column. Can be used with str and num data. If there is more than one such value, only the smallest is return - 'constant': replace missing values with fill_value. Can be used with strings or numeric data
Fill_value	Str or numerical value. It is used when strategy == 'constant'

Statistical Imputation

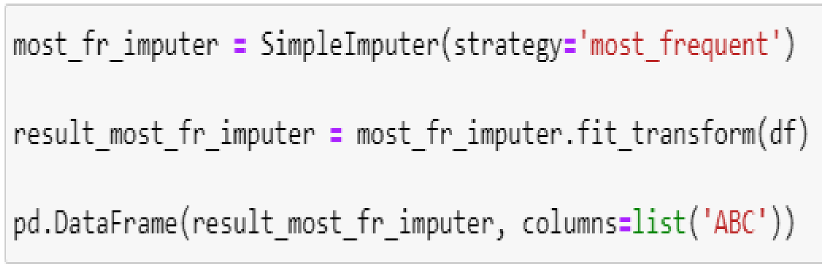
SimpleImputer method



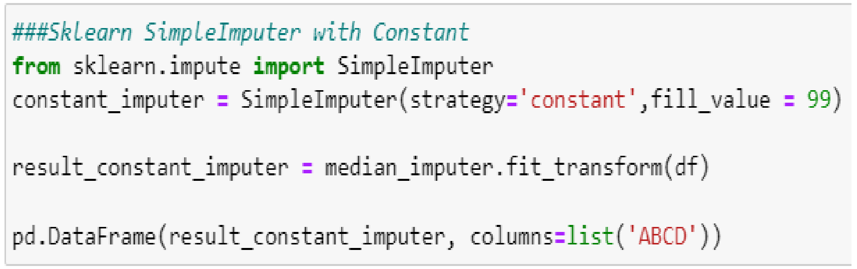
```python
###Sklearn SimpleImputer with Mean
from sklearn.impute import SimpleImputer
mean_imputer = SimpleImputer(strategy='mean')

mean_imputer.fit(df)

# pd.DataFrame(result_mean_imputer, columns=list('ABCD'))
```



```python
median_imputer = SimpleImputer(strategy='median')

result_median_imputer = median_imputer.fit_transform(df)

pd.DataFrame(result_median_imputer, columns=list('ABC'))
```



```python
most_fr_imputer = SimpleImputer(strategy='most_frequent')

result_most_fr_imputer = most_fr_imputer.fit_transform(df)

pd.DataFrame(result_most_fr_imputer, columns=list('ABC'))
```



```python
###Sklearn SimpleImputer with Constant
from sklearn.impute import SimpleImputer
constant_imputer = SimpleImputer(strategy='constant',fill_value = 99)

result_constant_imputer = median_imputer.fit_transform(df)

pd.DataFrame(result_constant_imputer, columns=list('ABCD'))
```

KNN Imputation

KNNImputer method

Parameters	Description
missing_values	The placeholder for the missing values. It can be int, float, str, np.nan, None or pandas.NA, default = np.nan
n_neighbors	Int, default 5
weights	{‘uniform’, ‘distance’} or callable, default = ‘uniform’. Weight function used in prediction - ‘uniform’: all points in each neighborhood are weighted equally - ‘distance’: weight points by inverse of their distance - Callable: a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights
metric	Distance metric for searching neighbors, default = ‘nan_Euclidean’

KNN Imputation

KNNImputer method

- NaN Euclidean Distances
  - Calculates the Euclidean distances in the presence of missing values
  - The formulation ignores feature coordinates with a missing value in either sample and scales up the weight of the remaining coordinates

- Formula

Dist(x,y) = sqrt(weight * sq. distance from present coordinates)

Where

Weight = total no of coordinates / no of present coordinates

- Example
  - X = [3,None,None,6]
  - Y= [1,None,4,5]
  - Nan_Euclidean_distance(X,Y) = 3.16227766

Outlier Identification and Removal

Standard Deviation Method

- The Gaussian distribution has the property that the standard deviation from the mean can be used to reliably summarize the percentage of values in the sample:
  - 1 Standard Deviation from the Mean: 68 percent
  - 2 Standard Deviations from the Mean: 95 percent
  - 3 Standard Deviations from the MEAN: 99.7 percent
- A value that falls outside of 3 standard deviations is part of the distribution, but it is a rare event at approximately 1 in 370 samples.
- Three standard deviations from the mean is a common cut-off in practice for identifying outliers in a Gaussian or Gaussian-like distribution

Outlier Identification and Removal

Standard Deviation Method

```
# identify outliers with standard deviation
from numpy.random import seed
from numpy.random import randn
from numpy import mean
from numpy import std
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# calculate summary statistics
data_mean, data_std = mean(data), std(data)
# define outliers
cut_off = data_std * 3
lower, upper = data_mean - cut_off, data_mean + cut_off
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
print('Identified outliers: %d' % len(outliers))
# remove outliers
```

```
...
# remove outliers
outliers_removed = [x for x in data if x > lower and x < upper]
```

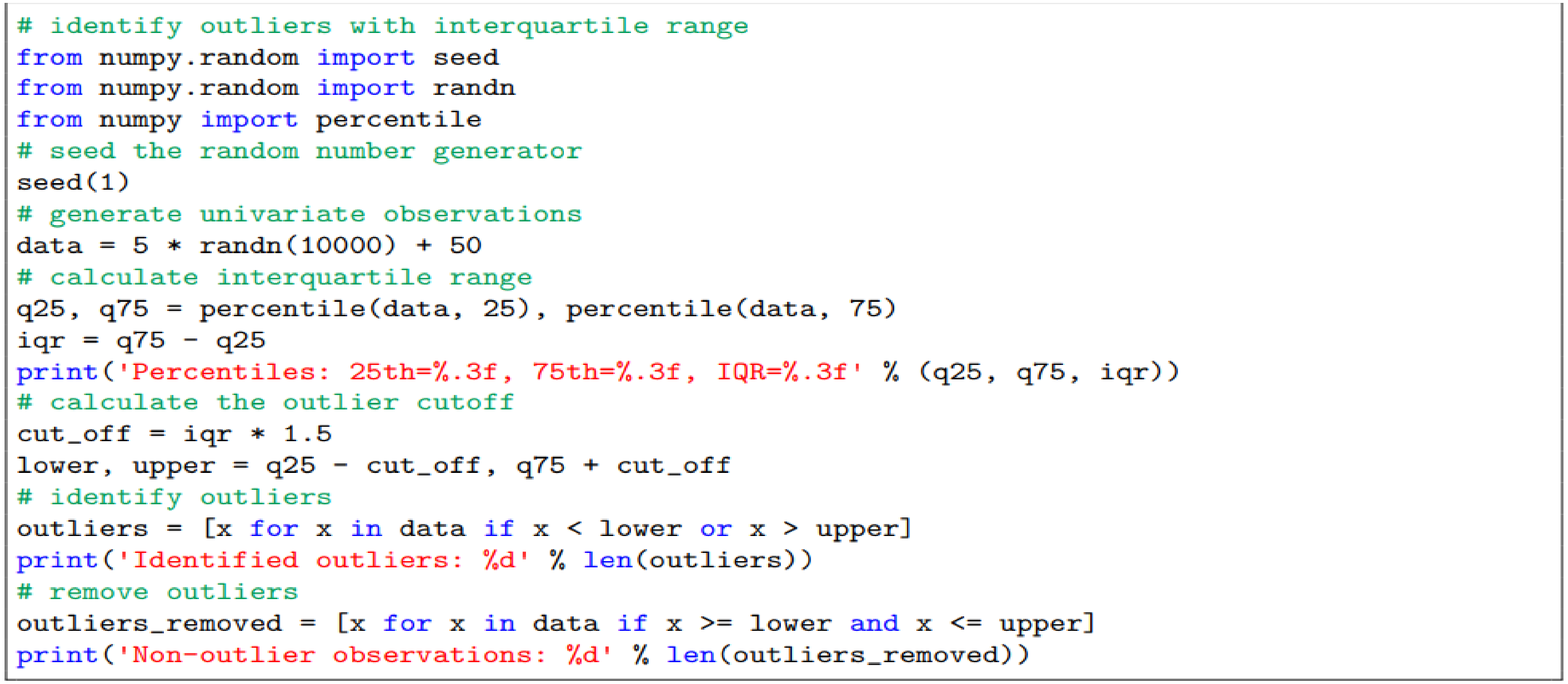
Outlier Identification and Removal

Interquartile Range Method

- Not all data is normal or normal enough to treat it as being drawn from a Gaussian distribution
- A good statistic for summarizing a non-Gaussian distribution sample of data is the Interquartile Range or IQR for short
- The IQR is calculated as the difference between the 75th and the 25th percentiles of the data
- The IQR can be used to identify outliers by defining limits on the sample values that are a factor k of the IQR below the 25th percentile or above the 75th percentile
- The common value for the factor k is the value 1.5

Outlier Identification and Removal

Interquartile Range Method



```
# identify outliers with interquartile range
from numpy.random import seed
from numpy.random import randn
from numpy import percentile
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# calculate interquartile range
q25, q75 = percentile(data, 25), percentile(data, 75)
iqr = q75 - q25
print('Percentiles: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75, iqr))
# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
print('Identified outliers: %d' % len(outliers))
# remove outliers
outliers_removed = [x for x in data if x >= lower and x <= upper]
print('Non-outlier observations: %d' % len(outliers_removed))
```