Pandas

Mercer 2018

Pandas

Pandas is a Python library designed for working with data. It provides functions and methods to deal with common data analysis problems.

Basic data structures

- Series
- DataFrame

Series

A Series is a one-dimensional object similar to an array, a list, or a column in a table. Each item in Series has an index. By default, each item will receive an index label from 0 to N, where N is the length of the Series minus one.

In [1]:
```
import pandas as pd
import numpy as np

pd.Series([1, 'abc', 4.3, 'another item'])
```

Out[1]:
```
0               1
1             abc
2             4.3
3    another item
dtype: object
```

Series

Index can also be specified manually. For example:

In [2]:
```
pd.Series([29, 37, 31, 32], index=[2015, 2016, 2017, 2018])
```

Out[2]:
```
2015    29
2016    37
2017    31
2018    32
dtype: int64
```

Series

Alternatively, you can also create a Series from dictionary, by specifing index: value pairs.

In [3]:
```
obs = {2015: 29, 2016: 37, 2017: 31, 2018: 32}
pd.Series(obs)
```

Out[3]:
```
2015    29
2016    37
2017    31
2018    32
dtype: int64
```

Subsetting Series

Subseting series

You can select values from the series using index:

In [4]:
```
gdp = pd.Series(dict(United_States=19390600, China=12014610, Japan=4872135, Germany=3684
816,
           United_Kingdom=2624529, India=2611012))
```

In [5]:
```
gdp['India']
```

Out[5]:
```
2611012
```

Subseting series

Or using logical operators as filters (so called boolean mask):

In [6]:
```
gdp >= 4872135
```

Out[6]:
```
United_States      True
China              True
Japan              True
Germany           False
United_Kingdom    False
India             False
dtype: bool
```

In [7]:
```
gdp[gdp >= 4872135]
```

Out[7]:
```
United_States    19390600
China            12014610
Japan             4872135
dtype: int64
```

Basic operations

Basic mathematical operations are possible both using scalars and functions.

In [8]:
```
gdp * 2
```

Out[8]:
```
United_States     38781200
China             24029220
Japan              9744270
Germany            7369632
United_Kingdom     5249058
India              5222024
dtype: int64
```

Basic operations

In [9]:

```
import numpy as np
np.sqrt(gdp)
```

Out[9]:

```
United_States     4403.475900
China             3466.209746
Japan             2207.291326
Germany           1919.587456
United_Kingdom    1620.039814
India             1615.862618
dtype: float64
```

Basic operations

Operations on two Series are conducted using the index, that is calculations are performed on series joined by index.

In [10]:
```
population = pd.Series({'China': 1384.7, 'India': 1296.8, 'United_States': 329.3, 'Japa
n': 126.2})
gdp_per_capita = gdp / population
gdp_per_capita
```

Out[10]:
```
China              8676.688091
Germany                    NaN
India              2013.426897
Japan             38606.458003
United_Kingdom             NaN
United_States     58884.300030
dtype: float64
```

If the corresponding index can't be found in one of the Series, NaN is returned.

Basic operations

Series have also built-in methods. For example you can find NaN values using `isnull()` method.

In [11]:
```
gdp_per_capita.isnull()
```

Out[11]:
```
China             False
Germany            True
India             False
Japan             False
United_Kingdom     True
United_States     False
dtype: bool
```

In [12]:
```
gdp_per_capita[gdp_per_capita.isnull()]
```

Out[12]:
```
Germany          NaN
United_Kingdom   NaN
dtype: float64
```

Basic operations

If we want to find non-null values we can use `notnull()` method.

In [13]:
```
gdp_per_capita[gdp_per_capita.notnull()]
```

Out[13]:
```
China             8676.688091
India             2013.426897
Japan            38606.458003
United_States    58884.300030
dtype: float64
```

Or just negate `isnull()` with `~`.

In [14]:
```
gdp_per_capita[~gdp_per_capita.isnull()]
```

Out[14]:
```
China             8676.688091
India             2013.426897
Japan            38606.458003
United_States    58884.300030
dtype: float64
```

Basic operations

We can get a total of all elements using `sum()` method.

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

Out[15]:
```
45197702
```

A list of all Series methods can be found [here (https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.Series.html)](https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.Series.html).

DataFrames

DataFrame

A DataFrame comprises of rows and columns, and is similar to database table, R's data.frame object or Excel's spreadsheet. It is useful to think of a DataFrame as a group of Series objects that share an index.

Creating DataFrame

You can create DataFrame from a dictionary in a similar fashion to creating Series, but in this case a key is a given column's name and values are column's content.

In [16]:
```
pd.DataFrame({'country': ['United_States', 'China', 'Japan', 'Germany', 'United_Kingdom'
, 'India'],
              'gdp': [19390600, 12014610,  4872135,  3684816,  2624529,  2611012],
              'population': [329.3, 1384.7, 126.2, np.nan, np.nan, 1296.8]})
```

Out[16]:

	country	gdp	population
0	United_States	19390600	329.3
1	China	12014610	1384.7
2	Japan	4872135	126.2
3	Germany	3684816	NaN
4	United_Kingdom	2624529	NaN
5	India	2611012	1296.8

Creating DataFrame from file

Usually, you will not create a DataFrame manually, but load it from some source. Pandas is quite flexible in this regard, and allows you to read from many sources including csv files, sql queries, Excel files, url and so on.

In [17]:
```
iris = pd.read_csv('../data/iris.csv')
iris.head()
```

Out[17]:

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

Creating DataFrame from clipboard

Sometimes it is useful to create DataFrame on the fly from your clipboard. One of the use cases is getting content from Excel or sql query, without the need to set up connection etc. To do that, simply copy data to your clipboard and then run `read_clipboard()` function.

Excercise 1 (3 min)

1. Read data from this url (https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv) without downloading it.
2. Try to do that again using a clipboard.

Initial data exploration

Initial data exploration

To get a better understanding of the data at hand you can use `info()` and `describe()` methods. `info()` presents data regarding columns, their types, a number of non null values, as well as a memory footprint.

In [18]:
```
titanic = pd.read_csv('../data/titanic.csv')
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

Describe method

`describe()` on the other hand shows basic descriptive statistics regarding the DataFrame.

In [19]:

```
titanic.describe()
```

Out[19]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Describe method

It is sometimes easier to analyse the result of `describe()` method with columns and rows switched. You can do this by adding `T` (transpose method) preceeded with the comma after `describe()`.

In [20]:
```
titanic.describe().T
```

Out[20]:

	count	mean	std	min	25%	50%	75%	max
PassengerId	891.0	446.000000	257.353842	1.00	223.5000	446.0000	668.5	891.0000
Survived	891.0	0.383838	0.486592	0.00	0.0000	0.0000	1.0	1.0000
Pclass	891.0	2.308642	0.836071	1.00	2.0000	3.0000	3.0	3.0000
Age	714.0	29.699118	14.526497	0.42	20.1250	28.0000	38.0	80.0000
SibSp	891.0	0.523008	1.102743	0.00	0.0000	0.0000	1.0	8.0000
Parch	891.0	0.381594	0.806057	0.00	0.0000	0.0000	0.0	6.0000
Fare	891.0	32.204208	49.693429	0.00	7.9104	14.4542	31.0	512.3292

Additional tools

Additional tools for getting to know your data better can be found in `pandas_profiling` package. It allows you to generate html with some basic as well as more advanced info.

```
import pandas_profiling as pf

profile = pf.ProfileReport(titanic)
profile.to_file(outputfile="profile.html")
```

Head and tail

Two other useful methods, that give a quick glimpse of the data, are `head()` and `tail()`. First displays a couple (5 by default) top rows from the DataFrame. `tail()` works in similar fashion, but takes rows from the bottom.

In [21]:
```
titanic.head()
```

Out[21]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Columns attribute

Sometimes it is useful to acces column names as a list of strings. You can use `DataFrame` `columns` attribute for that.

In [22]:
```
titanic.columns
```

Out[22]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

It is useful if you want to change names of columns.

In [23]:
```
titanic_2 = titanic.copy()
titanic_2.columns = list(range(len(titanic.columns)))
titanic_2.head()
```

Out[23]:

	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Rename method

Columns' names can be also changed with `rename()` method.

In [24]:
```
titanic_2 = titanic_2.rename(axis = 'columns', mapper = {1: 'one', 2: 'two'})
titanic_2.head()
```

Out[24]:

	0	one	two	3	4	5	6	7	8	9	10	11
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Excercise 2 (15 min)

Your friend is concerned with incoming alien invasion and has asked you to help him analyze data that he has collected regarding ufo sightings. Do some preliminary analysis.

1. Load csv file from data folder (`ufo.csv`).
2. How many columns there are in the dataset?
3. How many rows?
4. How many of rows have missing values?
5. What are the data types?
6. Calculate basic descriptive statistics.
7. Generate profile report for the data set.

Selecting rows and columns

Column selection

As mentioned previously, it is useful to think of a DataFrame as a collection of Series. Taking that into consideration, it should come as no suprise that selecting a single column from the DataFrame will return a Series object.

In [25]:
```
titanic['Survived'].head()
```

Out[25]:
```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

In [26]:
```
type(titanic['Survived'])
```

Out[26]:
```
pandas.core.series.Series
```

Column selection

Please note that you can refer to columns both using `[]` and `.`

In [27]:
```
titanic.Name.head(3)
```

Out[27]:
```
0                              Braund, Mr. Owen Harris
1    Cumings, Mrs. John Bradley (Florence Briggs Th...
2                               Heikkinen, Miss. Laina
Name: Name, dtype: object
```

In [28]:
```
titanic['Name'].head(3)
```

Out[28]:
```
0                              Braund, Mr. Owen Harris
1    Cumings, Mrs. John Bradley (Florence Briggs Th...
2                               Heikkinen, Miss. Laina
Name: Name, dtype: object
```

Column selection

Selecting multiple columns returns DataFrame.

In [29]:
```
titanic[['Survived', 'Sex']].head()
```

Out[29]:

	Survived	Sex
0	0	male
1	1	female
2	1	female
3	1	female
4	0	male

Column selection

You can also return single column as DataFrame using double brackets (`[[ ]]`).

In [30]:
```
titanic[['Survived']].head()
```

Out[30]:

	Survived
0	0
1	1
2	1
3	1
4	0

In [31]:
```
type(titanic[['Survived']])
```

Out[31]:
```
pandas.core.frame.DataFrame
```

Rows selection

Selecting rows can be done in similar fashion as in Series by using boolean mask.

In [32]:
```
titanic[titanic.Survived == 1].head(2)
```

Out[32]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

Which could be translated to: show all records from titanic where titanic column Survived equals to 1.

Rows selection

Multiple conditions can be chained together using \& as **and**:

In [33]:
```
titanic[(titanic.Survived == 1) & (titanic.Age < 18)].head(2)
```

Out[33]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S

| for **or**:

In [34]:
```
titanic[(titanic.Age > 90)|(titanic.Age < 1)].head(2)
```

Out[34]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
78	79	1	2	Caldwell, Master. Alden Gates	male	0.83	0	2	248738	29.00	NaN	S
305	306	1	1	Allison, Master. Hudson Trevor	male	0.92	1	2	113781	151.55	C22 C26	S

Query method

Another option to filter rows of interest is to use DataFrame's `query()` method. The unusual feature of this method is that you pass the condition as a string. For example, if we were to translate previous filter to `query()` method we would receive the following:

In [35]:
```
titanic.query('Age > 90 or Age < 1').head(2)
```

Out[35]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
78	79	1	2	Caldwell, Master. Alden Gates	male	0.83	0	2	248738	29.00	NaN	S
305	306	1	1	Allison, Master. Hudson Trevor	male	0.92	1	2	113781	151.55	C22 C26	S

Notice that you do not have to specifiy the DataFrame in the condition as in case of previous operations (we don't need to write `titanic.query("df['Age']>90...")`).

Setting index

Alternatively, you can use index to select data. Our current index consists of row numbers. We can make it more meaningful by changing it to Name using `set_index()` method.

In [36]:
```
titanic_with_name_index = titanic.set_index('Name')
```

In [37]:
```
titanic_with_name_index.tail(3)
```

Out[37]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
Name											
Johnston, Miss. Catherine Helen "Carrie"	889	0	3	female	NaN	1	2	W./C. 6607	23.45	NaN	S
Behr, Mr. Karl Howell	890	1	1	male	26.0	0	0	111369	30.00	C148	C
Dooley, Mr. Patrick	891	0	3	male	32.0	0	0	370376	7.75	NaN	Q

Index attribute

You can access index with `DataFrame` `index` attribute.

In [38]:
```
titanic_with_name_index.index
```

Out[38]:
```
Index(['Braund, Mr. Owen Harris',
       'Cumings, Mrs. John Bradley (Florence Briggs Thayer)',
       'Heikkinen, Miss. Laina',
       'Futrelle, Mrs. Jacques Heath (Lily May Peel)',
       'Allen, Mr. William Henry', 'Moran, Mr. James',
       'McCarthy, Mr. Timothy J', 'Palsson, Master. Gosta Leonard',
       'Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)',
       'Nasser, Mrs. Nicholas (Adele Achem)',
       ...
       'Markun, Mr. Johann', 'Dahlberg, Miss. Gerda Ulrika',
       'Banfield, Mr. Frederick James', 'Sutehall, Mr. Henry Jr',
       'Rice, Mrs. William (Margaret Norton)', 'Montvila, Rev. Juozas',
       'Graham, Miss. Margaret Edith',
       'Johnston, Miss. Catherine Helen "Carrie"', 'Behr, Mr. Karl Howell',
       'Dooley, Mr. Patrick'],
      dtype='object', name='Name', length=891)
```

Loc and iloc methods

With index in place we can select passengers by name using `loc()` method (label-based indexing):

In [39]:
```
titanic_with_name_index.loc[['Behr, Mr. Karl Howell', 'Dooley, Mr. Patrick']]
```

Out[39]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
Name											
Behr, Mr. Karl Howell	890	1	1	male	26.0	0	0	111369	30.00	C148	C
Dooley, Mr. Patrick	891	0	3	male	32.0	0	0	370376	7.75	NaN	Q

If we would like to select rows by position (positional indexing), we could use `iloc()` method:

Loc and iloc

In case of both `iloc()` and `loc()` methods we can also specify a subset of columns. For example:

In [41]:
```
titanic_with_name_index.loc[['Behr, Mr. Karl Howell', 'Dooley, Mr. Patrick'], ['Fare', 'Age']]
```

Out[41]:

Name	Fare	Age
Behr, Mr. Karl Howell	30.00	26.0
Dooley, Mr. Patrick	7.75	32.0

In [42]:
```
titanic_with_name_index.iloc[1:5, 4:8]
```

Out[42]:

Name	Age	SibSp	Parch	Ticket
Cumings, Mrs. John Bradley (Florence Briggs Thayer)	38.0	1	0	PC 17599
Heikkinen, Miss. Laina	26.0	0	0	STON/O2. 3101282
Futrelle, Mrs. Jacques Heath (Lily May Peel)	35.0	1	0	113803
Allen, Mr. William Henry	35.0	0	0	373450

So in both cases we first define rows that we are interested in and then columns: `df.(i)loc[rows, columns]`.

Excercise 3 (10 min)

Your friend is concerned that the government is trying to cover up something and believes that they are corrupting the data.

1. Remove a not meaningful column (hint: you can use `drop()` method).
2. Filter out incomplete observations from dataset.
3. Convert duration (seconds) and latitude columns into a float.
4. Find cities in Canada where UFO visited for more than 24 hours.
5. Find cities visitied by UFO in United Kingdom and Australia.
6. Display rows 3 to 6 and 10 to 15 with second, third and fourth column.

Creating new columns

Creating new columns

Creating new columns is similar to creating new key-value pairs in a dictionary. As in dictionary, you call the object with a new key and value. Consider the example below.

In [43]:
```
titanic['Fare_rounded'] = np.round(titanic.Fare)
titanic.head()
```

Out[43]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Fare_rounded
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	7.0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	71.0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	8.0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	53.0
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	8.0

When creating a new column you must use `df['column_name']` notation as `df.column_name` works only for exisiting columns.

Assign method

Alternatively, you can use `assign()` method to accomplish the same goal.

In [44]:

```
titanic = titanic.assign(Fare_rounded = lambda x: np.round(x.Fare))
titanic.head()
```

Out[44]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Fare_rounded
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	7.0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	71.0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	8.0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	53.0
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	8.0

Drop method

If you would like to delete one or more columns you can use `drop()` method.

In [45]:
```
titanic.drop('Fare_rounded', axis=1).head()
```

Out[45]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Dropping inplace

Both drop and assign return a new DataFrame. So in order to propagate the changes into our previous DataFrame, we have to reassign this new data frame to the old reference. Some of the Pandas methods have also `inplace` parameter, which if set to True will automatically change the old DataFrame. This could be done for `drop()` method:

In [46]:
```
titanic.drop('Fare_rounded', axis=1, inplace=True)
titanic.head()
```

Out[46]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Axis parameter

Another thing to notice regarding `drop()` is the `axis` parameter, which tells Pandas that we would like to drop a column (instead of row when axis=0).

Excercise 4 (5 min)

Create two new columns:

1. 'duration_hours' where visit duration will be expressed in hours.
2. 'city_shape' that will containe the name of the city and shape of UFO separated by '-'

Modifing columns

Modyfing columns

To modify a column you can simply assign different value to it. For example, if we would like to get rough estimate of passenger age in days we can do the following:

In [47]:
```
titanic['Age'] = titanic['Age'] * 365
titanic.head()
```

Out[47]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	8030.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	13870.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	9490.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	12775.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	12775.0	0	0	373450	8.0500	NaN	S

Replace method

If we want to target specific value in the column, we can use `replace()` method. We could for example replace male with 0 and female with 1 in Sex column by:

In [48]:

```
titanic.Sex.replace({'female': 1, 'male': 0}, inplace=True)
# titanic.Sex.replace(['female', 'male'], [0, 1] , inplace=True) - we can also pass two lists
titanic.head()
```

Out[48]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	0	8030.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	13870.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	1	9490.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	12775.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	0	12775.0	0	0	373450	8.0500	NaN	S

Numpy where function

Alternatively we could use `np.where()` which has perhaps a more familiar syntax, similar to excel's if statement.

In [49]:
```
titanic = pd.read_table('https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv',
                        sep = ',') #we are reloading the dataset to erase the implemented changes
titanic['Sex'] = np.where(titanic.Sex == 'male', 0, 1)
titanic.head()
```

Out[49]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	0	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	1	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	0	35.0	0	0	373450	8.0500	NaN	S

Apply method

So far we were modyfing one column at a time. We can also change multiple columns with `apply()` method which can be used to change either subset of columns or all of them. We can pass lambda function to it, describing what we want to do. We could for example standardize Age, SibSb and Fare columns following way:

In [50]:
```
titanic.loc[:, ['Age', 'SibSp', 'Fare']] = (titanic.loc[:, ['Age', 'SibSp', 'Fare']].
                                            apply(lambda x: (x - np.mean(x))/np.std(x)))
titanic.head()
```

Out[50]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	0	-0.530377	0.432793	0	A/5 21171	-0.502445	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	0.571831	0.432793	0	PC 17599	0.786845	C85	C
2	3	1	3	Heikkinen, Miss. Laina	1	-0.254825	-0.474545	0	STON/O2. 3101282	-0.488854	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	0.365167	0.432793	0	113803	0.420730	C123	S
4	5	0	3	Allen, Mr. William Henry	0	0.365167	-0.474545	0	373450	-0.486337	NaN	S

Fillna method

One more useful method to know is `fillna()` , which allows us to change missing values to some other value. For example, in some cases we are missing the Cabin info. We can fill it with the 'unknown' string:

In [51]:
```
titanic.Cabin.fillna('unknown').head()
```

Out[51]:
```
0    unknown
1        C85
2    unknown
3       C123
4    unknown
Name: Cabin, dtype: object
```

Excercise 5 (5 min)

1. Create new column 'is_us' where with True/False values indicating whether visit happend in the USA.
2. Change 'city' column, so cities' names begin with a upper-case letter.

Grouping and summarising data

Split-Apply-Combine

Split-Apply-Combine concept has been popularized by Hadley Wickham. In his own words:

> *Many data analysis problems involve the application of a split-apply-combine strategy, where you break up a big problem into manageable pieces, operate on each piece independently and then put all the pieces back together.*

Split-Apply-Combine

Usually we can extract groups of observations that we are interested in. For example we can analyse our data groupped by sex and age:

name	age	sex
John	13	Male
Mary	15	Female
Alice	14	Female
Peter	13	Male
Roger	14	Male
Phyllis	13	Female

.(sex)

name	age	sex
John	13	*Male*
Peter	13	*Male*
Roger	14	*Male*

name	age	sex
Mary	15	*Female*
Alice	14	*Female*
Phyllis	13	*Female*

.(age)

name	age	sex
John	*13*	Male
Peter	*13*	Male
Phyllis	*13*	Female

name	age	sex
Alice	*14*	Female
Roger	*14*	Male

name	age	sex
Mary	*15*	Female

Split-Apply-Combine

Summary statistics are then calculated based on defined groups. We can, for example, count ocurrences of different combinations of variables:

.(sex)

sex	value
Male	3
Female	3

.(age)

age	value
13	3
14	2
15	1

.(sex, age)

sex	age	value
Male	13	2
Male	14	1
Female	13	1
Female	14	1
Female	15	1

Groupby method

In Pandas `groupby()` method serves this purpose. It returns a DataFrameGroupBy object which has a variety of methods, many of which are similar to standard SQL aggregate functions.

In [52]:
```
titanic.groupby('Sex')
```

Out[52]:
```
<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x00000223F7AC1588>
```

Count and size methods

For example `count()` returns the total number of non null values in each column, while `size()` gives you total number of records.

```
In [53]: titanic.groupby('Sex').count()
```

Out[53]:

	PassengerId	Survived	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
Sex											
0	577	577	577	577	453	577	577	577	577	107	577
1	314	314	314	314	261	314	314	314	314	97	312

```
In [54]: titanic.groupby('Sex').size()
```

```
Out[54]: Sex
0    577
1    314
dtype: int64
```

Sort values method

We can use combination of `size()` and `sort_values()` which orders rows by specified value, to get top 5 Cabins by number of passengers.

In [55]:
```
titanic.groupby('Cabin').size().sort_values(ascending=False)[:5]
```

Out[55]:
```
Cabin
B96 B98        4
G6             4
C23 C25 C27    4
F2             3
C22 C26        3
dtype: int64
```

Value counts method

Alternatively, we could achieve the same with `value_counts()` method, which counts the occurences of each value in a Series:

In [56]:
```
titanic.Cabin.value_counts()[:5]
```

Out[56]:
```
B96 B98        4
C23 C25 C27    4
G6             4
E101           3
D              3
Name: Cabin, dtype: int64
```

Summary statistics

There are many built-in methods that you can use on grouped DataFrame. You can easily calculate basic summary statistics either on selected columns or all of them:

In [57]:
```
titanic.groupby('Sex')[['Age', 'Fare']].mean()
```

Out[57]:

Sex	Age	Fare
0	0.070784	-0.134506
1	-0.122855	0.247166

In [58]:
```
titanic.groupby('Sex').std()
```

Out[58]:

Sex	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
0	257.486139	0.391775	0.81358	1.011152	0.963422	0.612294	0.868575
1	256.846324	0.438211	0.85729	0.972019	1.049355	1.022846	1.167765

Transform method

You can also use your own functions using `transform()` function. It will apply function you passed to each group. Let's assume that we would like to calculate deviation from mean age of each Sex for each passenger.

In [59]:
```
titanic['age_deviation_by_sex']=titanic.groupby('Sex').Age.transform(lambda x: x - x.mea
n())
```

In [60]:
```
titanic.head()
```

Out[60]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	age_deviat
0	1	0	3	Braund, Mr. Owen Harris	0	-0.530377	0.432793	0	A/5 21171	-0.502445	NaN	S	-0.601161
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	0.571831	0.432793	0	PC 17599	0.786845	C85	C	0.694686
2	3	1	3	Heikkinen, Miss. Laina	1	-0.254825	-0.474545	0	STON/O2. 3101282	-0.488854	NaN	S	-0.131969
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	0.365167	0.432793	0	113803	0.420730	C123	S	0.488022
4	5	0	3	Allen, Mr. William Henry	0	0.365167	-0.474545	0	373450	-0.486337	NaN	S	0.294383

Agg method

Another useful method is `agg()`, which can take a dictionary, where key is column name that we would like to perform aggregation on, and values are types of aggregations. Let's assume we would like to know mean ticket prices for most expensive cabines, with the number of passenger that were allocated to them:

In [61]:

```
(titanic.groupby('Cabin').agg({'Fare': [np.mean,
                                         np.size]})
                         .sort_values(('Fare', 'mean'),
                                      ascending=False)[:5])
```

Out[61]:

	Fare	
	mean	size
Cabin		
B101	9.667167	1.0
C23 C25 C27	4.647001	4.0
B57 B59 B63 B66	4.634417	2.0
B51 B53 B55	4.559709	2.0
B58 B60	4.335332	2.0

Because our columns are now a MultiIndex, we need to pass in a tuple specifying how to sort.

Agg method

We can perform aggregation on multiple columns too. Let's add the median age of people in cabin and their mean survival rate to the previous example:

In [62]:
```
titanic.groupby('Cabin').agg({'Fare': [np.mean, np.size],
                              'Age': np.median,
                              'Survived': np.mean}).sort_values(('Fare','mean'), ascendi
ng=False)[:5]
```

Out[62]:

	Fare		Age	Survived
	mean	size	median	mean
Cabin				
B101	9.667167	1.0	0.365167	1.0
C23 C25 C27	4.647001	4.0	-0.427045	0.5
B57 B59 B63 B66	4.634417	2.0	-0.702597	1.0
B51 B53 B55	4.559709	2.0	0.330723	0.5
B58 B60	4.335332	2.0	0.502943	0.5

Excercise 6 (20 min)

Your friend has couple of questions:

1. How many different shapes of UFO there are? Which is the most common?
2. How many cities in each country has been visited by ufo?
3. What is the longest median duration of visit among cities with more than 10 visits?
4. In which country there was the longest UFO visit? Convert seconds to hours.

Digging deeper - pandas data types

Pandas data types

Pandas has a following data types:

- object - text values
- int64 - integer numbers
- float64 - floating point numbers
- bool - true/false values
- datetime64 - date and time values
- timedelta[ns] - difference between two datetimes
- category - categorical value

Pandas data types

The number after int and float indicates max value that it can contain. For example int8 can hold values from -128 to 127 (2^8, that is 256 possible values). It is sometimes useful to downsize values in DataFrame, when we know that they are unlikely to surpass max value for given type in order to lower the memory footprint.

Pandas data types

For example, in titanic dataset SibSp (number of siblings and spouses) contains values from 0 to 8:

In [63]:
```
titanic.SibSp.min(), titanic.SibSp.max()
```

Out[63]:
```
(-0.47454519624983954, 6.784163299176891)
```

Astype method

We can use `astype()` method to change type...

In [64]:
```
titanic_lower_memory = titanic.copy()
titanic_lower_memory['SibSp'] = titanic_lower_memory.SibSp.astype(np.int8)
```

In [65]:
```
np.round((titanic_lower_memory.memory_usage().sum())/(titanic.memory_usage().sum())- 1,
2)
```

Out[65]:
```
-0.07
```

...which allowed us to lower the memory usage of DataFrame by 7%.

Category data type

The **category type** uses integer values under the hood to represent values in a column, rather than the raw values. Pandas uses a separate mapping dictionary that maps the integer values to the raw ones. This arrangement is useful whenever a column contains a limited set of values. When we convert a column to the category dtype, Pandas uses the most space efficient int subtype that can represent all of the unique values in a column.

Category data type

Sex column contains only two values so it fits nicely into category data type

In [66]:
```
titanic['Sex'] = titanic.Sex.astype('category')
```

Accessor

An important point to remember is that data types have built-in methods that help in specific cases. They can be accessed through so called accessors. You can think of a Pandas' accessor as a property that serves as an interface to additional methods.

Cat accessor

Previously, we change Sex column to category and now we can access additional methods for this dtype by using **cat** accessor. We can for example see all available levels following way:

In [67]:
```
titanic.Sex.cat.categories
```

Out[67]:
```
Int64Index([0, 1], dtype='int64')
```

Cat accessor

... or rename levels:

In [68]:
```
titanic.Sex.cat.rename_categories({'male': 'm', 'female': 'f'}).head()
```

Out[68]:
```
0    0
1    1
2    1
3    1
4    0
Name: Sex, dtype: category
Categories (2, int64): [0, 1]
```

Other accessors

Pandas has also useful built-in method for working with text columns, which can be accessed through **str** accessor and for working with dates through **dt** accessor.

Str accessor

Str accessor provides variety of methods to deal with text columns. For example we could be interested in checking whether Name column contains title 'Mr.' which indicates gender of the passenger. We can achieve this with `contains()` method. We have to escape dot sign with backslash as dot is by default treated as regex sign that matches any character.

In [69]:
```
titanic.Name.str.contains('Mr\.').head()
```

Out[69]:
```
0     True
1    False
2    False
3    False
4     True
Name: Name, dtype: bool
```

Str accessor

Another useful method is `split()` which allows to create list of elements after split. We can chain it with `transform()` which will take only the first element of that list (surname in this case).

In [70]:

```
titanic.Name.str.split(',').transform(lambda y: y[0]).head()
```

Out[70]:

```
0       Braund
1      Cumings
2    Heikkinen
3     Futrelle
4        Allen
Name: Name, dtype: object
```

Dt accessor

Dt accessor on the other hand allows us to get more information from a date. It has variouse methods to extract different parts of the date - month, day, weekday etc. Let's assume that we would like to know how many alien spottings happend at the beginning of month...

In [71]:
```
ufo = pd.read_csv('../data/ufo.csv', low_memory=False)
ufo['datetime'] = pd.to_datetime(ufo.datetime, format='%d/%m/%Y %H:%M', errors='coerce')
```

In [72]:
```
ufo.datetime.dt.is_month_start.sum()
```

Out[72]:
```
2929
```

Dt accessor

...or extract month and weekday name:

In [73]:
```
ufo.datetime.dt.month.head()
```

Out[73]:
```
0    10.0
1    10.0
2    10.0
3    10.0
4    10.0
Name: datetime, dtype: float64
```

In [74]:
```
ufo.datetime.dt.weekday_name.head()
```

Out[74]:
```
0       Monday
1       Monday
2       Monday
3    Wednesday
4       Monday
Name: datetime, dtype: object
```

Excercise 7 (5 min)

Calculate what percent ufo sightings happend over the weekend.

iDash

mb@idash.pl mo@idash.pl